CentraSite

Implementation Concepts

Version 9.6

April 2014

Document ID: IINM-IG-DEPLOY-96-20140318

Table of Contents

Preface

This document describes concepts and considerations associated with implementing CentraSite for different use-case goals. This guide is intended for the individuals who are involved with the deployment and implementation of CentraSite, including the Enterprise Architects and Implementation Leaders who are responsible for planning and managing the implementation of CentraSite at their site.

⚠ **Important:** This guide provides suggested approaches and practices for implementing CentraSite. However, you should always test that a particular approach or recommendation to ensure that it satisfies the needs of your organization.

The content is organized under the following sections:

Overview of CentraSite	Provides a high-level overview of the major components in a CentraSite environment.
Implementation Decisions and Configuration Tasks	Identifies the high-level decisions that you need to make and tasks you need to complete when implementing CentraSite.
Obtaining and Installing the Starter Kit	Describes how to obtain and install a set of examples that you can reference while preparing for your implementation.
Choosing a Deployment Strategy	Describes the approaches you can use to implement CentraSite in terms of single- or multi-stage deployments.
Defining Your Organizational Structure	Describes the role of Organizations in a CentraSite registry and identifies issues you should consider when defining organizations for a CentraSite implementation.
Setting Up Users and Groups	Describes the role of Users and Groups in a CentraSite registry and identifies issues you should consider when defining organizations for a CentraSite implementation.
Using Permissions and Roles to Manage Access to the Registry	Describes how CentraSite uses permissions and roles to control access to registry objects and operations and identifies issues you should consider when assigning roles and permissions to CentraSite users.
Customizing Your Asset Catalog	Describes ways in which you can customize the registry to support the types of assets that make up your SOA environment and identifies issues you should consider when making these customizations.
Working with Versions and Revisions	Describes the versioning and revisioning capabilities in CentraSite and discusses when to generate new versions of existing assets.
Defining Lifecycle Models	Describes how CentraSite uses lifecycle models to enable governance of assets in the registry and identifies issues you should consider when applying lifecycle models to the assets in your SOA environment.
Defining Design/Change-Time Policies	Describes how you can use design/change-time policies to control the creation, modification and deletion of registry objects and identifies

	issues you should consider when applying design/change-time policies to your registry.
Planning Your Run-Time Environment	Provides an overview of the components that make up the run-time environment and describes how the components interoperate when CentraSite is implemented across two stages.
Implementing the Mediation Environment	Describes issues to consider when configuring webMethods Mediator for collecting performance metrics and logging. Also provides information about using CentraSite with Insight.
Managing Run-Time Policies	Describes how to use run-time policies to enforce security policies, perform logging tasks and monitor performance metrics for virtual services.
Managing Virtualized Services	Describes how to create virtual services, who should create virtual services and when to create virtual services. Also discusses which services you should consider virtualizing.
Identifying the Consumers of Virtual Services	Describes how consumer applications are identified at run time and are specified at design time.
Managing Endpoints	Describes how CentraSite represents the endpoints for services and how to keep endpoint information up-to-date as a service moves through its development lifecycle.

1 Overview of CentraSite

Today's enterprises are rapidly adopting Service Oriented Architecture (SOA) as a strategy for delivering business applications that can be developed and extended quickly.

CentraSite is a comprehensive and extensible SOA governance platform that you can deploy to support numerous usage scenarios. CentraSite provides a single platform for design-time and run-time governance, enabling enterprises to manage their Web services from the request stage through the deployment and maintenance stages of their lifecycle.

Design-Time Features and Benefits

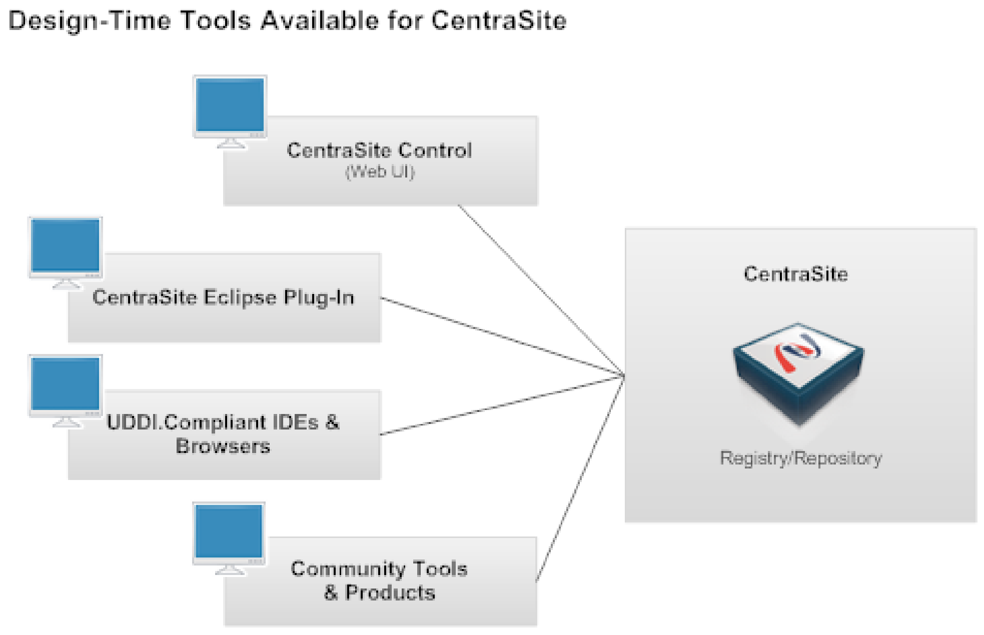
CentraSite supports the development of SOA-based applications by enabling developers, architects and business analysts to:

- Publish Web services and other reusable assets into their organization's central registry.
- Discover Web services and other reusable assets and use them to assemble new applications.
- Obtain detailed information about a Web service, including the list of its consumers, its technical support contacts, its disposition in the development lifecycle, usage tips and performance data.
- Examine the relationships that a Web service has with other artifacts in the SOA environment in order to understand how a change to that service will impact the service's sub-components and dependants.

CentraSite supports an array of design-time tools that enable developers, architects and business analysts to discover, publish and reuse SOA assets.

Managing the content of the registry is critical to the success of an SOA environment. To support this effort, CentraSite's governance capabilities and tools enable administrators and architects to:

- Control access to CentraSite and to individual assets listed in the registry.
- Model the specific entities and artifacts that make up an organization's SOA environment.
- Enable reuse of computing assets by providing easy access to in-depth information about an artifact's technical properties, semantics and relationships to other artifacts in the SOA.
- Define classification systems (taxonomies) to ensure that Web services and other assets can be easily discovered and managed.
- Impose mandatory testing, approval processes and/or quality checks on assets to ensure that they adhere to organizational standards and policies.
- Model the development lifecycle for a type of asset and specify policies that are to be triggered when an asset transitions from one lifecycle state to another.

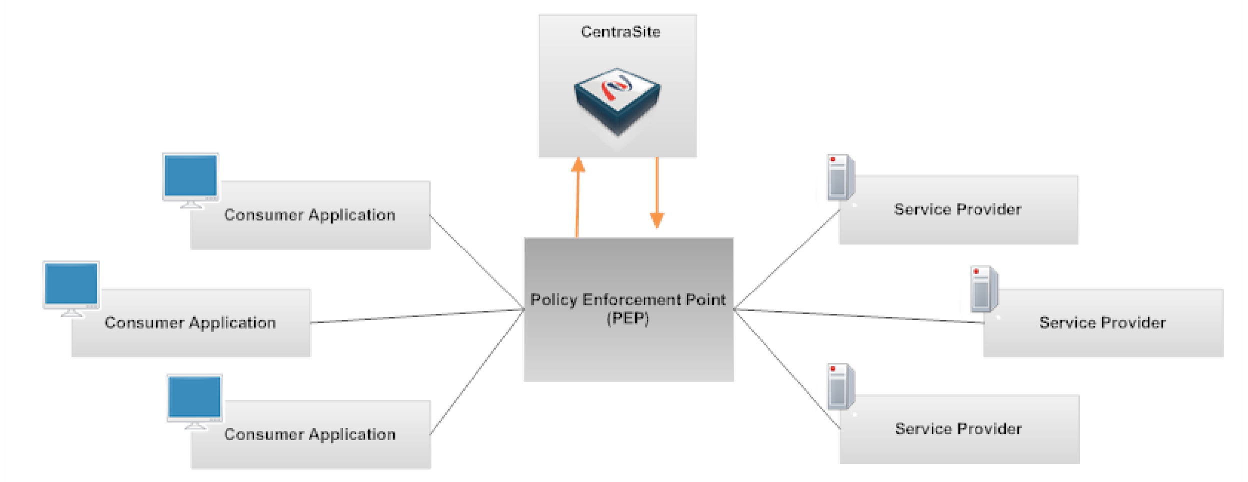

Design-Time Tools Available for CentraSite
CentraSite Control
(Web UI)
CentraSite Eclipse Plug-In
UDDI.Compliant IDEs & Browsers
Community Tools & Products
CentraSite
Registry/Repository

Run-Time Features and Benefits

CentraSite provides tools that support the management and monitoring of services in the run-time environment. Using CentraSite, administrators can define policies that execute on policy enforcement points (PEPs) that reside between the consumer and the service endpoint. These policies typically perform security-related activities (such as authentication and message encryption/decryption), auditing/logging tasks and performance reporting functions.

When webMethods Mediator is used as a policy enforcement point, administrators can define and deploy “virtual services” into the run-time environment. Virtual services operate as consumer-facing proxies for the endpoints where Web services are actually hosted. Besides performing security, logging and monitoring activities, a virtual service can also execute advanced mediation steps such as message routing, load-balancing, failover handling and message transformation.

A PEP sits between the Consumer and the Provider



CentraSite supports the run-time environment by enabling you to:

- Define and manage standard run-time policies.
- Attach run-time policies to Web services and deploy the policies to specified PEPs in the run-time environment.
- Define and deploy "virtual services" to perform mediation steps such as routing, load-balancing, failover and/or message transformation.
- Monitor the run-time performance of services and identify services that fail to meet specified thresholds.

Out of the box, CentraSite provides support for the following policy-enforcement points and run-time monitoring products:

- *webMethods Mediator*, which is a PEP that provides policy enforcement, service mediation and monitoring capabilities. webMethods Mediator enforces run-time policies that you create in CentraSite.
- *webMethods Insight*, which is a monitoring tool that enables you to see what is happening in real-time with service transactions as they flow across any system. It provides visibility and control at the transaction level to heterogeneous SOA environments.

CentraSite also includes a framework called *SOALink*, which enables it to interoperate with other, third-party PEPs.

2 Implementation Decisions and Configuration Tasks

This section describes decisions and issues you should consider while you are planning your implementation of CentraSite. It also contains a checklist that summarizes the key configuration and preparatory steps that must be completed during the implementation process.

Issues to Consider Before Implementation

Before you begin your implementation of CentraSite you should first define the goals and principles that you want to achieve by establishing SOA governance. Here are some of the questions you should think about beforehand:

- Who are the stakeholders and what organizations do you want to reach with your SOA-governance initiative?
- What are their roles in the development of SOA artifacts?
- What are the governance rules and processes that you want to establish?
- What kind of artifacts do you want to manage utilizing CentraSite?
- How can you support better reuse by introducing company-wide classification schemes for assets?
- How will you measure the success of your SOA environment and your SOA-governance initiatives?
- What is the development lifecycle of your artifacts today and what should it be? What type of stakeholder interactions occur during the lifecycle transitions?

Configuration Checklist

The following identifies the key configuration decisions and tasks that you will need to perform in order to prepare your implementation of CentraSite for use.

- Determine whether you will implement a single registry or multiple registries. See ***Choosing a Deployment Strategy***.
- Install each instance of CentraSite and configure its connection to the external authentication system. See ***Using CentraSite with an External Naming Directory***.
- Identify the organization structure you will use for each registry, identify organization administrators, and create the organizations. See ***Defining Your Organizational Structure*** and ***Setting Up Users and Groups***.
- Determine whether the default role assignments given to the Users group are appropriate for each organization and modify these assignments as necessary. See ***Consumer Organizations*** and ***Configuring the Default Roles that CentraSite Assigns to Users in an Organization.***

- Identify the types of assets that you want to catalog in CentraSite and define new types and/or customize the predefined types as necessary. Create the required taxonomies and/or association types to support your customizations. See ***Customizing Your Asset Catalog***.
- Determine which asset types you will place under lifecycle management and create the required lifecycle models. See ***Defining Lifecycle Models***.
- Determine which polices you want to place on the objects in your registry and create the required design/change-time policies to enforce these policies. See ***Defining Design/Change-Time Policies***.
- Create the Consumer Registration policy on each instance of CentraSite. See ***Using Policies to Execute a Consumer Registration*** and ***The Consumer Registration Policy***

If you will be using CentraSite with webMethods Mediator for run-time mediation, you must also perform the following tasks to prepare CentraSite for use:

- Define a target and a user account for each webMethods Mediator that is attached to an instance of CentraSite. See ***Implementing the Mediation Environment***.
- Create the lifecycle model and associated policies required to enable deployment of virtual services. See ***Defining the Lifecycle Model for Services and Virtual Services***.
- Define the process your site will use for creating, deploying, and promoting virtual services and make sure that the participants in this process have the necessary permissions to perform their assigned tasks. See ***Managing Run-Time Policies*** and ***Managing Virtual Services***.

3 Obtaining and Installing the Starter Kit

Software AG provides a starter kit for CentraSite that include examples of many of the registry objects that are described in this guide. You can install this kit to create a “demo” environment in which many of the concepts presented in this document have been implemented.

To obtain the starter kit, go to the **CentraSite Developer Community** and follow the installation instructions provided with the kit.

4 Choosing a Deployment Strategy

This section contains information you need to think about before deploying CentraSite in your environment.

Deploying CentraSite to Support Your SDLC

When developing Web services and other assets, most IT organizations follow a systems development life cycle (SDLC) that includes at least the following basic phases:

- *A development phase*, when Web services and other assets are requested and developed, individual contributions are integrated and development tests are conducted.
- *A test phase*, when services and other assets are tested in a controlled environment that mimics production scenarios.
- *A production phase*, when services and other assets are made operational. Generally, when an asset reaches this stage the environment in which it resides is tightly controlled and access to the asset itself is very restricted.

In CentraSite, the series of steps that make up your SDLC are represented by a *lifecycle model*. The lifecycle model is customized to your environment and enables you to establish governance controls over all phases of the SDLC for different types of assets.

When planning your CentraSite implementation, the first thing you must do is choose a deployment strategy that supports your organization's SDLC and its implementation requirements. The strategy you choose will determine the number of stages (instances of a CentraSite registry) that your organization will maintain and how it will map the phases of its SDLC to these stages.

Deployment Options

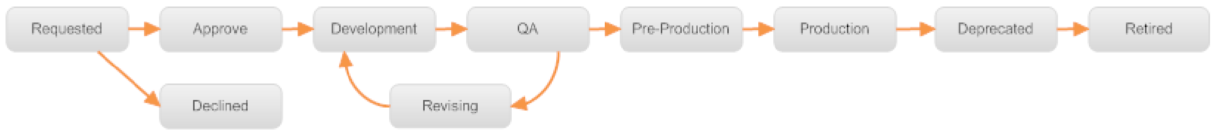
This section presents three basic deployment strategies and describes how you might map the phases of the SDLC to each of them. As a first step in planning your CentraSite implementation, you should review the following strategies and choose the strategy that best supports your organization's SDLC and its governance objectives.

- *Single-Stage Deployment:* In a single stage deployment, the entire SDLC is represented within one instance of CentraSite. With this strategy, you deploy and maintain a single registry. The assets that you place in the registry remain there over their entire lifecycle.

  When you use the single-stage strategy, you map all three basic phases of the SDLC to a single lifecycle model in CentraSite. To promote an asset through the SDLC (for example, to move it from the development phase to the test phase), you simply switch the asset's lifecycle state in the registry.

The following figure depicts a single-stage deployment. Note that the states that make up its lifecycle comprise the full SDLC (from request to retirement).

Single-Stage Deployment



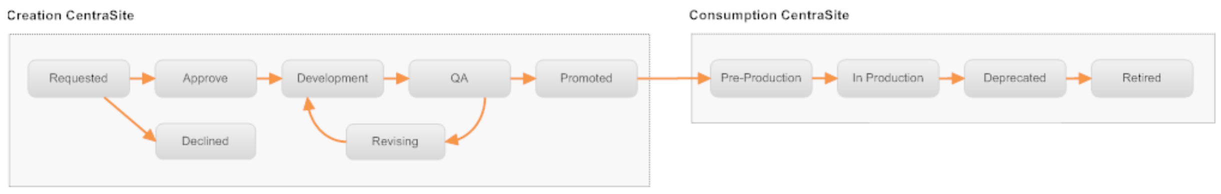
- *Two-Stage Deployment:* In a two-stage deployment, the SDLC is split between two instances of CentraSite. One instance, called the *creation CentraSite,* is used to manage assets during the development and test phases of the SDLC. The other instance, called the *consumption CentraSite,* manages assets that are in the production phase of the SDLC.

  This strategy enables your organization to completely separate assets in the pre-production phases from assets that are actually operational. (In some organizations, the physical separation of development and production systems is necessary to satisfy legal regulations.)

  When you use the two-stage approach, the SDLC is represented by two lifecycle models in CentraSite. One lifecycle model exists on the creation CentraSite. This model represents the states that make up the development and test phases of the SDLC. The other model exists on the consumption CentraSite. It represents the states that make up the production phase of the SDLC.

  To promote an asset to a phase of its lifecycle that resides on another stage, you export the asset from its current registry and import it into the registry that hosts the next phase of its lifecycle.

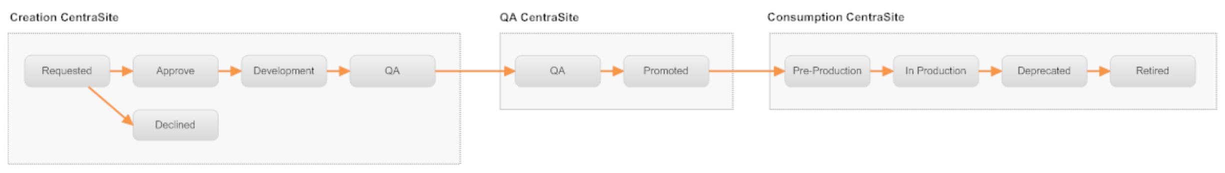
  Two-Stage Deployment



- *Three-Stage Deployment:* In a three-stage deployment, you deploy a separate registry for each major phase of the SDLC (Development, Test, Production).

  Each registry has a lifecycle model that represents the states that make up its phase of the SDLC. As with any approach that involves multiple stages, you promote an asset from one phase to the next by exporting the asset from its current registry and importing it into the registry that hosts the next phase of the SDLC.

  Three-Stage Deployment



Deployment Considerations

The deployment strategy you choose depends on factors such as your organization's policy requirements, standard processes and governance objectives. The following are points to consider when selecting a strategy for your organization:

- Realize that any deployment that involves multiple stages requires additional effort to configure and administer. Also keep in mind that the promotion process for a multi-stage environment will be more complex and time-consuming, because it involves physically exporting objects from one registry and importing them into another. You should not deploy a multi-stage configuration unless your organization has a compelling reason to do so. Aim for a deployment strategy that aligns well with your organization's SDLC process, satisfies your organization's governance objectives and uses the fewest number of stages.
- If you intend to use CentraSite for both design-time governance and run-time governance, consider using the two-stage deployment. This configuration enables you to maintain one registry for managing Web services (and associated assets) while they are in the development and testing phases of their lifecycle and another registry for configuring, deploying and monitoring Web services that are in the production phase of their lifecycle.

  Although it is possible to use a single-stage deployment for both design-time and run-time governance, such a configuration is suitable only for small or mid-size environments. Do not use a single-stage deployment if you intend to use CentraSite to manage both the design-time and run-time aspects of a large number of assets.

  Note: If you expect your organization's registry to begin small and grow over time, you are better off starting with a two-stage deployment rather than attempting to switch when you outgrow the single-stage configuration.

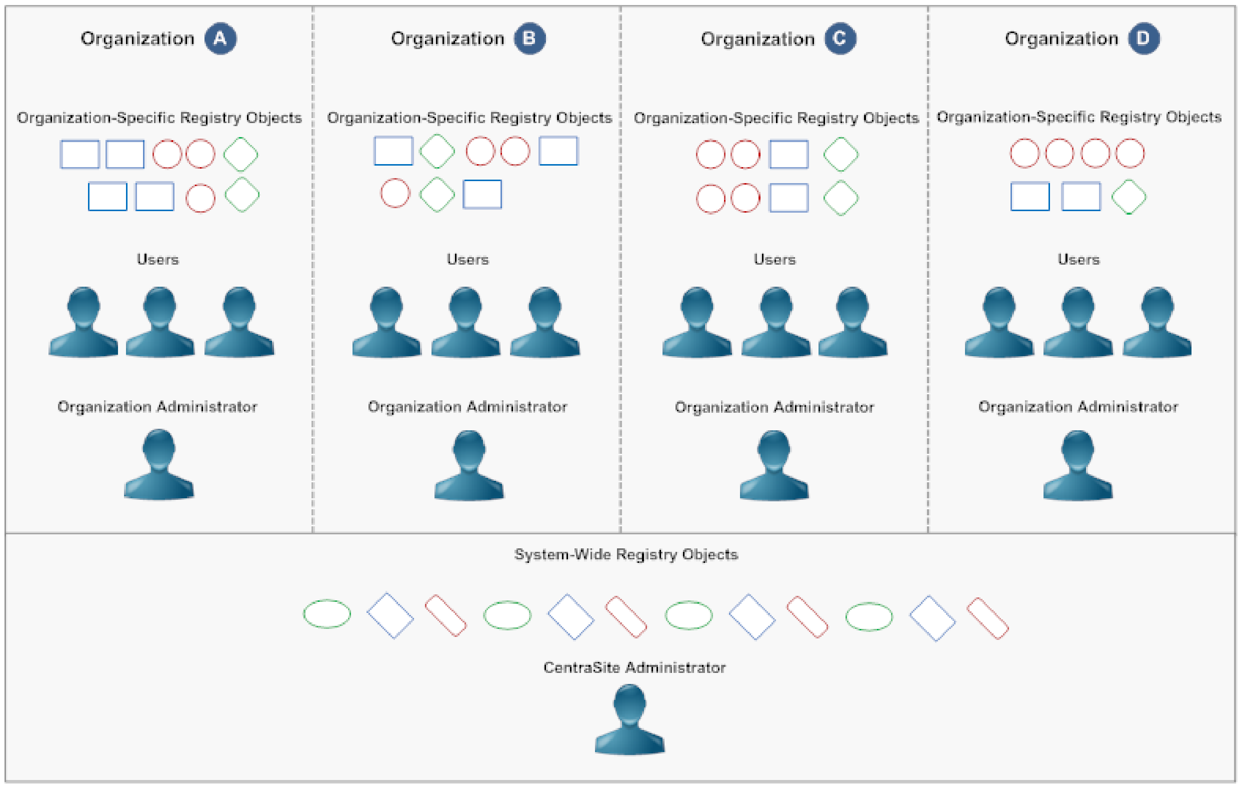
- If you will use CentraSite only for design-time governance, consider using a single-stage deployment. Deploying a multi-stage configuration for design-time governance does not generally offer any benefits. You should only consider a multi-stage deployment in a design-time implementation if your organization has a specific need to physically separate the registry of assets in the pre-production phases from the registry of assets in the production phase.

5 Defining Your Organizational Structure

After selecting your deployment strategy, you must define the organizational structure of your registry. The structure you choose determines how the assets in your registry are organized. It also plays a key role in controlling who can access various portions of the registry.

What is an Organization?

In CentraSite, an organization is a high-level container for the assets and other objects that make up a CentraSite registry. Any object that is not an organization must belong to an organization. You use organizations to partition your registry into autonomous collections of assets that you can administer independently. Typically, you use organizations to arrange your registry around functional lines of business, regional subsidiaries, branches, legal entities, departments, B2B partners (e.g., suppliers and customers) or similar criteria.


Organization A
Organization B
Organization C
Organization D
Organization-Specific Registry Objects
Users
Organization Administrator
System-Wide Registry Objects
CentraSite Administrator

By default, only users within an organization have permission to view the organization's assets. If other users require access to the organization's assets, they must obtain explicit permissions to do so. Thus, on a broad level, organizations enable you to restrict the visibility of a collection of assets to a specified group of users.

Organizations also act as a scoping mechanism for the following registry functions:

- Role-based permissions

- Lifecycle models
- Design/change-time policies
- Run-time policies
- Supporting documents

For example, organization administrators can create lifecycle models and design/change-time policies that apply only to the assets that belong to their organization.

An organization also represents a specific group of users within the registry. This quality allows you to apply many permissions and capabilities collectively to all users in an organization.

The Default Organization

CentraSite is installed with one predefined organization called Default Organization. The default organization owns the system-defined registry objects that CentraSite uses. You cannot delete the default organization nor can you rename it.

As a best practice, you should avoid using the default organization as an ordinary organization. Instead, treat it as the "home" for system-wide objects such as asset types, taxonomies, targets and system-wide policies, and restrict membership in this organization to a small number of administrative users.

Child Organizations

You can use parent-child associations to represent hierarchical relationships among organizations. Parent-child relationships enable you to collectively administer certain aspects of the associated organizations. For example, after you establish a parent-child relationship between organizations, administrators can collectively grant roles and permission to users that belong to a parent organization or any of its children. Moreover, the administrator of a parent organization also has administrative control over the organization's descendants (meaning he or she can manage users, assets, policies, lifecycles and so forth within the parent organization and all of its descendants).

To create a parent-child relationship, you must create the child organization from within the parent organization. A child organization functions just like an ordinary organization. It does not inherit any characteristics from its parent. It has its own administrator (although the administrator of the parent organization also has administrative privileges in the child organization) and it has its own set of users, policies, lifecycle models and assets. A child organization can have additional child organizations of its own.

As you define the organizational structure for your registry, identify organizations that you want to relate in a hierarchical form and use parent-child relationships to reflect the hierarchy.

When defining your organizational structure, keep the following points in mind with respect to parent-child relationships:

- A child organization can belong to only one parent.
- You must create the parent organization first and then create the child from the parent. In other words, you cannot create two unrelated organizations and then, at a later time, establish a parent-child relationship between them.
- Once you associate organizations in a parent-child relationship, you cannot disassociate the organizations.
- You cannot move a child organization to a different parent.
- You cannot promote a child organization to become top-level parent organization.
- A child does not inherit properties or objects from its parent (other than the fact that organization administrators in the parent organization are automatically allowed to administer the child organization). Nor do the parent's organization-specific policies and lifecycles apply to its children. A child organization creates and maintains its own policies and lifecycles independent of its parent. In this respect, it is like any other organization. (To impose policies and lifecycles on all organizations in the registry, you can create system-wide policies and lifecycles as described later in this guide.)

Consumer Organizations

By default, CentraSite gives every user in an organization the ability to act both as an asset consumer (meaning the user can view assets in the registry) and an asset provider (meaning the user can publish assets into the registry). CentraSite does this by assigning both the Asset Provider role and the Asset Consumer role to the organization's Users group (the group comprising all users in the organization).

If you have groups of users that will only consume assets (a group of external partners, perhaps), consider creating a separate organization for just those users. Remove the Asset Provider role from the organization's Users group so that the users in the organization have just the Asset Consumer role. Any organization that wants to extend assets to these consumers can do so by giving the consumer organization's Users group permission to view the assets.

Modeling Your Organizations

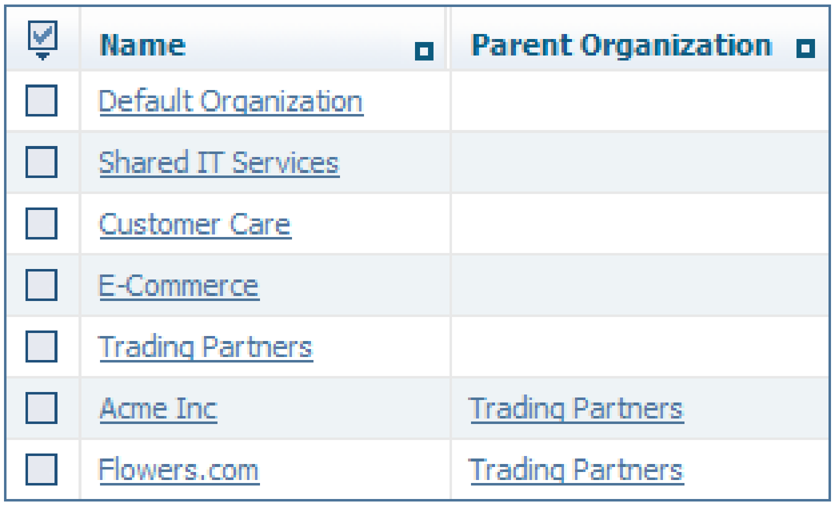
Although you might be tempted to create an organizational structure that precisely mimics the organization chart within your enterprise, this is not usually a practical strategy. Many low-level organizational units within your enterprise have no relationship to your SOA environment and, therefore, do not need to be represented in the registry. Furthermore, the low-level work groups within an enterprise tend to be very dynamic, which makes them unsuitable entities on which to base your registry's organizational structure.

Instead of organizing the contents of your registry around the low-level departments on an organization chart, group it by higher-level concepts such as functional units, lines of business, process owners, legal entities (e.g., subsidiaries and affiliates) or regional divisions. Think in terms of who owns the assets that will reside in the registry or has primary responsibility for developing and maintaining them. Create organizations to represent those areas.

If you intend to give external partners access to CentraSite, *create separate organizations for each partner*. (By partners we mean any external entities with which your enterprise interacts, such as suppliers and other vendors, dealers and distributors, customers, government agencies, trade organizations and so forth.)

The following shows the organizational structure that is defined in the starter kit example. This example has a level of granularity that is appropriate for defining organizations.

Example Organization Structure



✓	Name	Parent Organization
☐	Default Organization	
☐	Shared IT Services	
☐	Customer Care	
☐	E-Commerce	
☐	Trading Partners	
☐	Acme Inc	Trading Partners
☐	Flowers.com	Trading Partners

Organization	Description
Default Organization	This organization serves as the home for administrative and system-wide artifacts.
Shared IT Services	This organization represents Information Technology (IT) departments that operate across (i.e., are shared by) other lines of business. This organization would include the Quality Assurance (QA) department, the SOA Competency Center and individuals who serve as enterprise-wide IT architects.
Customer Care	This organization is responsible for systems and services related to customer management (from CRM to claims processing).
E-Commerce	This organization represents the department responsible for providing external-facing services including customer-facing portal sites and business-to-business services.
Trading Partners	This organization is the parent organization for all external partner organizations. It is a “pseudo” organization that is used to hold organizations that represent external parties.
Acme Inc.	This organization represents an external partner that provides or consumes assets. *This organization is a child of the Trading Partners organization.*
Flowers.com	This organization represents an external partner that provides or consumes assets. *This organization is a child of the Trading Partners organization*

If you are using a multi-stage deployment, you might replicate the same organizational structure across all registries. Or you might adopt a different structure in each registry. For example, on the creation CentraSite you might use the organizational structure like the one described above. However, on the consumption CentraSite, you might define just two organizations: an “operations organization”, which owns and manages all of the assets in the registry, and a “consumer organization”, which contains users who can browse the consumption registry. The organizational structure that you adopt for other stages will depend on your particular requirements.

Choosing an Organizational Strategy

Aside from choosing a deployment strategy, defining your organizational structure is one of the most critical deployment decisions you will make. It is important to choose a structure that is stable and will endure over time. After you establish your registry's structure and put your governance processes in place, it is difficult to make fundamental changes to the way in which the registry is organized. Such a change would not only require you to transfer assets to different organizations, but might also require you to redefine the lifecycle models, policies and permissions that support your governance environment.

When planning your organizational strategy, take the following points into consideration:

- In general, create organizations around the concept of asset visibility, ownership or responsibility. In the pre-production stages, use organizations to represent the major stakeholders involved in the pre-production aspects of the asset's development lifecycle (e.g., service owners, developers and the SOA Competency Center). In the production stage, use organizations to represent the groups of users who represent assets owners, consumers of assets and the operators of the production services.

- If a particular group of users requires the use of custom lifecycle models or design-time policies, create a separate organization for those users.
- If you have groups of users whose needs are "consumer only", create separate consumer organizations for those users.
- Keep in mind that a user can belong to only one organization within a CentraSite registry. If you have a user who will create assets for multiple organizations, add the user to the organization in which he or she will work primarily. Then use roles to give the user the ability to create assets in other organizations.
- Keep in mind that an asset also belongs to only one organization. To make an asset accessible to multiple organizations, you must give the users in those organizations permission to access the asset.
- Avoid creating an organizational structure that is too fine-grained. Keeping the structure of the registry synchronized with your low-level development teams and work-units will require a significant amount of work. Moreover, a fine-grained structure is generally not needed in order to govern your assets effectively.

6 Setting Up Users and Groups

Users represent individuals that are known to CentraSite. You assign roles and permissions to users to specify which operations they can perform and which registry objects they can access. (Roles and permissions are discussed in more detail later in this guide.) Groups enable you to collectively assign roles and permissions to groups of users.

Adding Users to CentraSite

To interact with a CentraSite registry, an individual must have a user account with the registry.

A user account is represented by an instance of a User object. A User object contains basic attributes such as the name, email address and phone number for an individual. It also includes a link to the individual's user account in an external authentication system. This attribute is required if the User object represents an individual who will actually log on to CentraSite. You can customize the User object type to include additional attributes as necessary.

A user account can be activated or deactivated by an administrator. Active users are allowed to log on to CentraSite. Inactive users exist in the registry, but they are not permitted to log on to CentraSite. Additionally, permissions cannot be granted to inactive users, inactive users cannot be assigned ownership of an asset (although they will retain ownership of the assets that they already own), nor can they be assigned to groups. Administrators generally deactivate users who leave the company or otherwise cease to be valid users of the registry. If you want to delete a user from CentraSite, you must deactivate the user first.

Tip: To keep your audit trail intact when a user leaves the registry, simply deactivate that user and leave his or her existing assets in place. If you delete the user or transfer the user's assets to someone else, the audit trail for those assets will be lost.

You can also use inactive users to represent individuals who are actors within your SOA environment, but are not actual users of the registry. For example, you might model certain line-of-business managers as “users” in the CentraSite registry so that you can express associations between these individuals and various assets in the registry. Such users might never log on to CentraSite themselves, and therefore would not need to have an account that is active and linked to the external authentication system. However, the registry would know of these users, so assets could be associated with them. (Points-of-contact for external parties such as suppliers and distributors are additional individuals that you might want to model as inactive users.)

Note: Because inactive users cannot be assigned to groups, users who are inactive are not eligible to receive automatic email notifications from CentraSite. You might want to take this point into consideration when deciding whether to make a user inactive or active.

Only users who belong to a role that includes the "Manage Users" permission can add, modify and/or delete users on CentraSite. However, any user (including guests) can view other users of the registry.

Using CentraSite with an External Naming Directory

Although CentraSite maintains its own database of user accounts, it authenticates users externally, either through the local operating system or an external directory service such as Active Directory or a Lightweight Directory Access Protocol (LDAP) server. Because authentication is handled by an external facility, it is not necessary for CentraSite to maintain its own set of user passwords. Password management is handled by your organization's existing authentication system.

By default, CentraSite is configured to authenticate users against the local operating system. While this configuration is adequate for initial experimentation and demonstration, it is generally not suitable for an enterprise-wide implementation of CentraSite. When you deploy CentraSite for actual use within your enterprise, you will need to configure it to authenticate users against a production-quality authentication service such as Active Directory or an LDAP server. Moreover, you should complete this configuration step *before* you begin creating organizations and setting up users, groups and roles.

Loading User Metadata from the External Directory

When you configure CentraSite to use Active Directory or LDAP for user authentication, you map the user metadata from the authentication system to the User object in CentraSite. This enables CentraSite to import the metadata (e.g., name, phone number, email address) for a user from the external directory when you create an account for that user in CentraSite. (Be aware that this is a one-time import. CentraSite simply loads the appropriate metadata from the authentication system into its database. It does not attempt to keep the user metadata synchronized with the authentication system afterwards.)

When you create user accounts in CentraSite, you can define the accounts individually or you can use the “bulk load” facility to import multiple users from the authentication system at one time.

Bootstrap Users, Organization Administrators and Primary Contacts

When you install an instance of CentraSite, it initially has two user accounts: an account for the *bootstrap user* and an account for the *default user*.

- *The bootstrap user* refers to the user who installs CentraSite. This user is given a user account in the Default Organization and becomes the initial Organization Administrator and Primary Contact for that organization. This user is also given the CentraSite Administrator role, which gives him or her “super admin” privileges. You can choose to assign each of these roles to other users later in the deployment process.
- *The default user* represents an internal user that owns the predefined objects installed with CentraSite. The default user exists for CentraSite's internal use. You cannot edit or delete this account. You cannot use the default user account to log on to CentraSite.

Note: There are actually a few additional user accounts that CentraSite creates for its own internal use, but the account for the default user is the only “visible” account that you or your users are likely to encounter while using CentraSite.

Generally, the bootstrap user creates the initial set of organizations. However, other users can perform this task if the bootstrap user adds those users to CentraSite and gives them the CentraSite Administrator role.

When you create an organization, CentraSite requires you to identify the users that will serve as the organization's *administrator* and as the organization's *primary contact*.

- *Organization Administrator*. The organization administrator is a user that has the Organization Administrator role for the organization. An organization must have at least one user in the Organization Administrator role. It can have multiple users in this role. A user in one organization can serve as an organization administrator for another organization; however, this role is usually given to someone within the organization. An organization administrator performs administrative tasks for the organization, such as:
  - Adding users to the organization
  - Defining groups and roles
  - Defining custom lifecycle models for the organization
  - Creating child organizations

  An Organization Administrator can also view, edit and delete any asset, policy or lifecycle model that belongs to his or her organization or to any of the organization's descendants.
- *Primary Contact*. The primary contact is simply a user who acts as the point-of-contact for an organization. An organization has just one primary contact. The user who is designated as the primary contact does not receive any additional roles or permissions by serving in this capacity. You can optionally select the same user to serve as both the administrator and the primary contact for an organization, but CentraSite does not require you to do this. You can have different users serving in each of these capacities.

After an organization is created, and its organization administrator and primary contact are assigned, the organization administrator (or any user with “Manage Users” permission for the organization) can begin adding additional users to the organization.

Guest Users

CentraSite supports the concept of a *guest user*. Guests are users who can access the registry without a user account (i.e., they access the registry anonymously).

The capabilities given to a guest are determined by the set of permissions specified in the Guest role. Typically, you give guest users read-only access to a limited set of objects.

By default, guests are given permission to use the “asset catalog” pages in the CentraSite Control user interface. They can use these pages to browse the assets whose permissions extend to the system-defined group called “Everyone”.

When you deploy CentraSite, think about the level of access you want to extend to guest users and configure the Guest role accordingly. Additionally, make sure that users who publish assets to your registry know that the Everyone group includes guest users, and that by granting access to this group, they enable access by anonymous users.

As a best practice, you should never give anything other than view permission to the Everyone group. This group should not be given permission to modify or delete registry objects.

Issues to Consider When Adding Users to CentraSite

When adding users to CentraSite, keep the following points in mind:

- To access a CentraSite registry in any capacity other than as a guest, a user must have a user account on CentraSite, and that account must be associated with a user account in an external authentication system.
- Users belong to organizations. You must create your organizations first and then add users to them.
- A user can belong to one and only one organization.
- As a general rule, users work with assets and objects that belong to their organization. However, an administrator can give users permissions to perform work in other organizations when necessary.

Defining and Using Groups in CentraSite

A group represents a specified set of users. A group can be empty or contain any number of users. It can contain users from different organizations.

Only administrators with “Manage Users” permission can create, edit and delete groups, however, all users (including guests) can view the groups that exist within an instance of CentraSite.

Ways in Which CentraSite Uses Groups

Within CentraSite, groups are used for the following purposes:

- To assign roles to groups of users.
- To give a group of users access to a specific object in the registry.
- To identify the group of individuals who are authorized to approve certain types of requests.
- To identify the target audience for certain policy actions (for example, the intended recipients of an email action).

System-Defined Groups Available in CentraSite

CentraSite provides the following system-defined groups.

- *Users*—The set of users that belong to an organization. Every organization has a Users group.
- *Members*—The set of users that belong to an organization or any of its descendant organizations. Every organization has a Members group.
- *Everyone*—The set of *all users* that are defined within an instance of CentraSite. *This group includes guest users.*

CentraSite manages the membership of these groups automatically. You cannot delete the system-defined groups or edit their membership. You can, however, edit the roles that are assigned to a system-defined group and use them in all of the same ways as you can a regular user-defined group.

Using Groups from Your External Authentication System

CentraSite can use groups that are defined in the external authentication system. When you use an external group with CentraSite, the membership of the group is defined and managed by the authentication system, not by CentraSite (i.e., you cannot use CentraSite to add members to the group or delete members from the group).

When CentraSite executes a request that references an external group, it accesses the external authentication system to resolve the group's membership. It performs the requested activity for each user who is a member of the specified group and is also a registered user on CentraSite. Users that are named in the external group but are not registered CentraSite users are ignored.

You can use externally defined groups in exactly the same way as native groups that you define in CentraSite. For example, you can assign roles to externally defined groups and you can grant permissions to them.

If your authentication system already defines groups of users that are significant to your SOA environment (e.g., SOA Architects, SOA Project Review Team, SOA Managers), consider adding them to CentraSite as external groups. Doing this will simplify maintenance by eliminating the need to update two systems when the membership of a group changes.

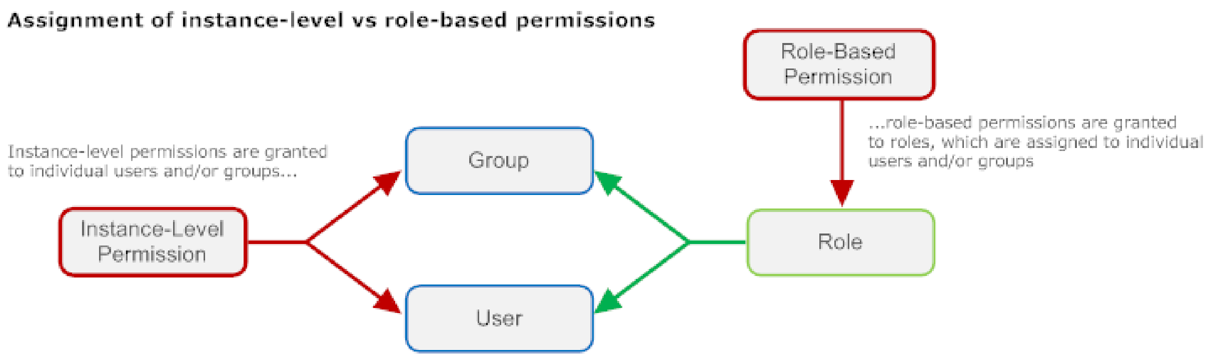
Note: Groups that are nested in the external authentication are supported by CentraSite. (If you are using LDAP, note that only the “recurse up” option is supported for group resolution. The “recurse down” option is not supported.)

7 Using Permissions and Roles to Manage Access to the Registry

Permissions determine which operations users can perform and which set of objects they can access. There are two types of permissions in CentraSite: *instance-level permissions* and *role-based permissions*.

- *Instance-level permissions* enable access to one specific instance of an object in the registry. They provide fine-grain access control over registry objects. You grant instance-level permissions directly to individual users or to groups.
- *Role-based permissions* enable access to an entire class of objects or give users the ability to perform certain operations in CentraSite. Role-based permissions provide coarse-grain control over objects in the registry. You assign role-based permissions to roles, and you assign the roles to individual users or groups.

The following diagram illustrates the relationships between instance-level permissions, role-based permissions, users, groups and roles.


Assignment of instance-level vs role-based permissions
Role-Based Permission
...role-based permissions are granted to roles, which are assigned to individual users and/or groups
Instance-level permissions are granted to individual users and/or groups...
Group
Instance-Level Permission
Role
User

Instance-Level Permissions

An instance-level permission gives a specified user or group access to one particular object in the registry. Instance-level permissions, which are granted at the View, Modify or Full level, determine how a specified user or group is allowed to interact with a registry object.

This Permission Level...	Enables the specified user or group to...
View	■ Access the object ■ Read the object's metadata (except its instance-level permission settings)
Modify	■ Perform all of the activities granted by the View permission ■ Edit the object's metadata (except its instance-level permission settings) ■ Read the object's instance-level permission settings
Full	■ Perform all of the activities granted by the View and Modify permissions

This Permission Level...	Enables the specified user or group to...
	■ Delete the object ■ Set the object's instance-level permission settings

Note that each permission level inherits the capabilities of the preceding level.

Objects that Support Instance-Level Permissions

Access control at the instance-level is not supported by all object types in the registry. The following list identifies the types of objects on which you can control access at the instance level.

- Assets (of any type)
- Supporting documents
- Design/Change-time policies **
- Run-time policies **
- Taxonomies **
- Report Templates **

** All CentraSite users have implicit and irrevocable permission to view all instances of these object types. However, you can use instance-level permissions to restrict who can edit and delete them.

Access to other types of objects is controlled using the broader role-based permissions (described below) or is enabled contextually. (Enabled contextually means that an object is a constituent of some other “access-controlled” object. For example, the individual operations and bindings associated with a Web service are objects that can only be accessed within the context of the Service object itself. Therefore, the permissions that control access to the Service object also control access to the service's constituent objects.)

Setting Instance-Level Permissions

To set permissions on an object, you must have Full permission on the object. You can use the CentraSite Control user interface to assign instance-level permissions to any of the types of objects listed above. Additionally, you can set permissions on the following types of objects using the CentraSite plug-in for Eclipse: assets, taxonomies, report templates and supporting documents.

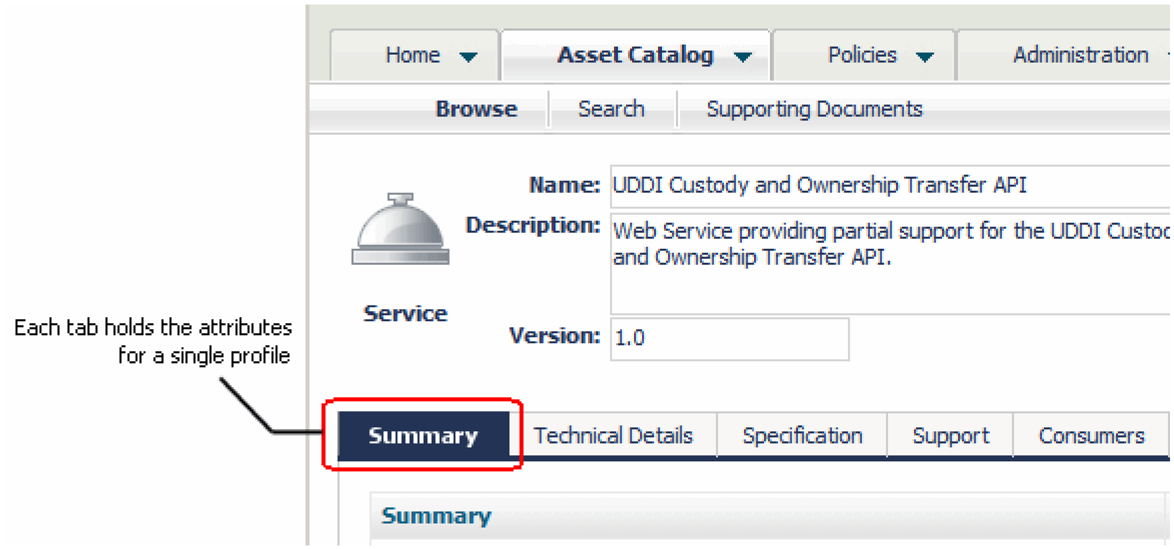
Besides setting instance-level permissions through the user interface, you can create design/change-time policies to automate the assignment of instance-level permissions on certain types of objects (specifically, assets and policies). For example, you might use a design/change-time policy to automatically extend access to specified groups of consumers when an asset switches to the “Deployed” state.

Profile Permissions

Assets (of any type) include an additional level of access control called a *profile permission*. Profile permissions enable you to control access to individual *profiles* within an instance of an asset.

A profile represents a collection of attributes. It is used to group the metadata for an asset when the asset is displayed in the user interface. Profiles enable CentraSite Control and the CentraSite plug-in for Eclipse to present the details for an asset in an organized manner. In CentraSite Control, for example, all of the attributes associated with a particular profile are grouped together on a separate tab.

The attributes associated with a profile are displayed on individual tabs in CentraSite Control



Name: UDDI Custody and Ownership Transfer API

Description: Web Service providing partial support for the UDDI Custoc and Ownership Transfer API.

Version: 1.0

Profile permissions determine which profiles a user sees when he or she views an asset with CentraSite Control or the CentraSite plug-in for Eclipse. You might use profile permissions, for example, to limit the amount of information that consumers see for an asset.

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.

Role-based Permissions

A role-based permission is a permission that CentraSite confers to users through a role. A role is a set of role-based permissions that you assign to users or groups.

Role-based permissions enable access to an entire set of objects or give users the ability to perform certain operations in CentraSite. Unlike instance-level permissions, which are granted directly to individual users or groups, role-based permissions are assigned to roles, and roles are assigned to users and groups. You cannot assign a role-based permission to a user or group directly.

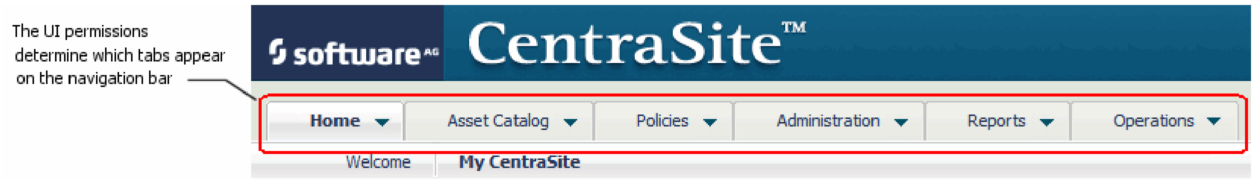
There are two basic types of role-based permissions:

- Permissions that enable access to areas of the CentraSite Control user interface (UI permissions)
- Permissions that enable users to create and/or access certain types of registry and repository objects (object-access permissions)

Permissions that Enable Access to Areas of the User Interface

Role-based permissions include a set of permissions that enable access to certain features and screen sets in the CentraSite Control user interface. These permissions determine which tabs are displayed to a user in CentraSite Control's navigation bar.

For example, CentraSite will not display the **Policies** tab to a user unless the user has the "Use the Policy UI" permission. The UI-related permissions also include permissions that enable users to view certain logs and use certain controls in CentraSite Control.



By default, all CentraSite users (including guests) have permission to use the Asset Catalog area of the CentraSite Control user interface. To use the other parts of the user interface, a user must belong to a role that *explicitly* includes the appropriate user interface permission (meaning that the role includes the actual permission itself) or *implicitly* includes the permission (meaning that the role includes a permission that implicitly grants the UI permission). For example, when you give a user permission to "Manage Design/Change-Time Policies", that permission implicitly grants that user permission to the "Use the Policy UI".

Permissions that Enable Access to Objects in the Registry or Repository

Role-based permissions also include permissions that enable users to create and/or work with an entire class of objects. Generally speaking, these types of role-based permissions grant a specified level of access to all objects of a specific type. For example, the *Modify Assets* permission grants *Modify* level access on all objects of type Asset. The role-based permissions enable you to apply access controls over an entire class of objects instead of assigning permissions on each instance of an object individually.

These role-based permissions are granted at varying levels. The name of the permission itself indicates the level of access that it grants.

If the name includes the following term...	The permission enables users to...
View	Read objects of a specified type. This level is equivalent to giving a user View instance-level permission on all objects of a given type.
Modify	Read and edit objects of a specified type. This level is equivalent to giving a user Modify instance-level permission on all objects of a given type.
Create	Create and read objects of a specified type. This level is equivalent to giving a user View instance-level permission of all objects of a given type and giving them the ability to create new instances of that type.
Manage	Create, read, edit, delete and modify the instance-level permission of objects of a specified type and modify the object's instance-level permissions. This level is equivalent to giving a user Full instance-level permission of all objects of a given type.

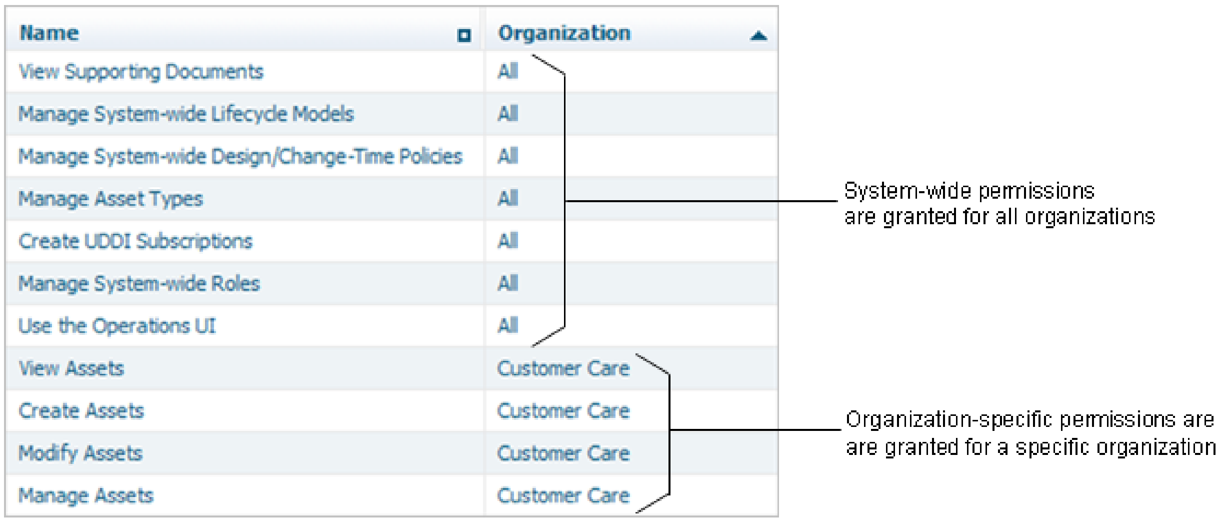
Note: Be aware that CentraSite does not provide role-based permissions at all levels for all object types. Access to certain objects types can only be granted at the Manage level, for example. Refer to the CentraSite user documentation for the role-based permissions available for each object type.

Organization-Specific vs. System-Wide Permissions

The permissions that enable access to registry/repository objects are either *organization-specific* or *system-wide*. An organization-specific permission grants a specific level of access to all objects of a given type *within a specified organization*. Permissions that enable access to assets, policies and lifecycle models are organization-specific. A system-wide permission grants access to objects that are available to all organizations, such as taxonomies and asset types.

System-wide permissions are generally given only to a small group of high-level administrators.

Permissions are system-wide or organization-specific



Name	Organization
View Supporting Documents	All
Manage System-wide Lifecycle Models	All
Manage System-wide Design/Change-Time Policies	All
Manage Asset Types	All
Create UDDI Subscriptions	All
Manage System-wide Roles	All
Use the Operations UI	All
View Assets	Customer Care
Create Assets	Customer Care
Modify Assets	Customer Care
Manage Assets	Customer Care

System-wide permissions are granted for all organizations

Organization-specific permissions are are granted for a specific organization

Issues to Consider when Working with Permissions

CentraSite provides you with coarse-grain and fine-grain permission controls. You will use both types, depending on your needs. When working with permissions in CentraSite, keep the following points in mind:

- All users, *including guests*, have implicit and irrevocable View permission on the following types of objects. You can control who can manage these types of objects (i.e., create, modify and delete them), but you cannot revoke view-level access to these objects.
  - Organizations
  - Users
  - Groups
  - Roles
  - Taxonomies
  - Asset Types
  - Policies
  - Run-Time Targets
  - Report Templates
- When a user has multiple instance-level permissions for the same object, the user receives the *union* of the combined permissions. For example, if a user has View permission on object ABC and belongs to a group that has Full permission on object ABC, the user will get Full permission on the object.

- When a user has both role-based permission and instance-level permission on the same object, the user also receives the *union* of the combined permissions. For example, if a user has the View instance-level permission on asset and also has the role-based "Manage Assets" permission, the user receives Full permission on the object (as conferred by the "Manage Assets" permission).
- There will be times when you need to decide whether to use instance-based or role-based permissions to grant a group of users access to a set of assets. When deciding which type of permission to use, keep the following points in mind:
  - When you grant role-based permissions to a group of users, those users are given permission to access *all assets* within the organization. You will not be able to selectively hide assets from certain members of the group. Additionally, all users in the group receive a specified level of access (i.e., View, Modify or Manage). You cannot selectively reduce this level of access for individual users. (Although you can selectively increase an individual user's level of access.)
  - If you grant access to an asset using a role-based permission, you will not be able to selectively hide *profiles* from the users with the role-based permission. The role-based permissions automatically confer permission to view all profiles for an asset.
  - Any approach that involves instance-level permissions, by definition, requires you to configure permissions on each asset individually. If you use instance-level permissions to routinely grant permissions to specified groups of users, consider creating a policy to do this for you automatically.
- If you need to hide or reveal certain profiles as an asset progresses through its lifecycle states, consider creating policies to automatically set the appropriate profile permissions when the asset switches state.
- Avoid granting instance-level permissions to individual users unless you have a specific reason to do so. Granting these permissions to groups instead of individuals, gives you greater flexibility and makes permission changes easier to manage.
- Be aware that if you grant instance-level permissions to an external group (i.e., a group that is defined and managed in your external authentication system), it might take CentraSite longer than normal to remove those permission assignments from a registry object.

Roles

A role is a set of role-based permissions. Assigning a role to a user gives the user the permissions specified in the role. Roles can be assigned to individual users or to groups. CentraSite is installed with several predefined roles, including the following:

- *CentraSite Administrator*. This role includes every role-based permission available in CentraSite. As such, it confers “super admin” capabilities to the users to which it is assigned. Users in this role can view, edit and delete virtually any object in the registry. This role cannot be deleted or edited. At least one user must be assigned to this role at all times.

- *Organization Administrator*. This role includes the set of role-based permissions that enable a user to administer an organization. Users in this role can view, edit or delete any object within an organization or the organization's descendants. This role cannot be deleted or edited. Each organization must have at least one user assigned to the organization's Organization Administrator role.
- *Asset Provider*. This role includes the "Create Assets" permission, which enables a user to create assets within his or her organization. By default, all users in an organization receive this role (however, you can configure this behavior as described later). You can modify the permissions associated with this role.
- *Asset Consumer*. This role includes the "View Assets" permission, which enables a user to view all of the assets that belong to his or her organization. By default, all users in an organization receive this role (however, you can configure this behavior as described later). You can modify the permissions associated with this role.

The predefined roles are usually adequate for most CentraSite implementations. However, you can create custom roles if the ones that CentraSite provides do not suit your needs. You can also modify many of the predefined roles that CentraSite supplies.

Note: Avoid creating a role that is equivalent to the CentraSite Administrator role. The CentraSite Administrator role is specifically optimized to maximize performance. An equivalent role will not perform as efficiently as the predefined CentraSite Administrator role that is installed with CentraSite.

To create or modify a role, you must have the "Manage Users" permission for the organization to which the role belongs. Be aware, however, that you cannot create a role that has more permissions than what your own user account has. For example, if you do not have the "Manage Taxonomies" permission, you cannot create a role that includes the "Manage Taxonomies" permission.

Assigning Roles to Users

You assign roles to users or groups using the CentraSite Control user interface. A user or group can be assigned multiple roles. When a user is given multiple roles, he or she receives the union of all permissions in those roles.

Configuring the Default Roles that CentraSite Assigns to Users in an Organization

By default, CentraSite assigns the Asset Provider and Asset Consumer roles to an organization's Users group. Consequently, every user that you add to an organization receives these roles.

If you do not want users in your organization to receive these roles automatically, or if you want to customize the set of permissions that users receive by default, you can do any of the following:

- Remove the Asset Provider and/or Asset Consumer roles from the organization's Users group.
- Modify the set of permissions associated with the Asset Provider and/or Asset Consumer roles that are assigned to the Users group.

- Create custom roles and assign them to the organization's Users group (instead of, or in addition to, the Asset Provider and/or Asset Consumer roles).

For example, if you want to create an organization whose users can only consume assets, you would remove the Asset Provider role from that organization's Users group. Doing this ensures that users added to the organization only receive permission to view assets. (An administrator could, of course, selectively give specific users in the organization permission to publish assets as necessary.)

8 Customizing Your Asset Catalog

CentraSite's flexible and extensible registry structure enables you to model virtually any kind of asset that you might want to include in your asset catalog. It supports a rich set of attribute types for defining the different properties and qualities of your assets. These types include attributes that you can use to classify an asset according to a predefined or custom taxonomy and attributes that you can use to associate the asset with other objects in the registry.

This section describes the three major aspects of your catalog that you can customize: *types*, *taxonomies* and *associations*.

Creating Custom Types

A *type* (also called an *object type*) describes a kind of object that the registry can store. Besides defining the set of attributes that make up an object, a type includes several system properties that determine, among other things, whether objects of the type are visible in the user interface, whether objects of the type can be used with reports and/or policies, and whether they can be versioned.

CentraSite includes many predefined types. You can customize many of these predefined types and also create custom types of your own.

Note: 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.

Object Types vs. Asset Types

All items stored in the CentraSite registry are *objects* of a particular type. Users, policies and taxonomies are examples of objects that are stored in the registry. An *asset* is a specific kind of object that represents an artifact in your SOA environment such as a Web service, an XML schema or a business process. In other words, all assets are objects, but not all objects are assets.

The *asset catalog* represents the set of all objects in your registry that are assets. Many features within CentraSite operate specifically on the contents of the asset catalog.

Any custom type that you add to CentraSite is considered to be an asset type. Consequently, all instances of a custom type are treated as assets.

Customizing the Predefined Asset Types Installed with CentraSite

CentraSite is installed with a number of predefined asset types, including types that represent Web services, XML schemas and BPEL processes. Before using these types in your environment, you should examine their type definitions and customize them as necessary.

With respect to customizing the predefined asset types installed with CentraSite, you can:

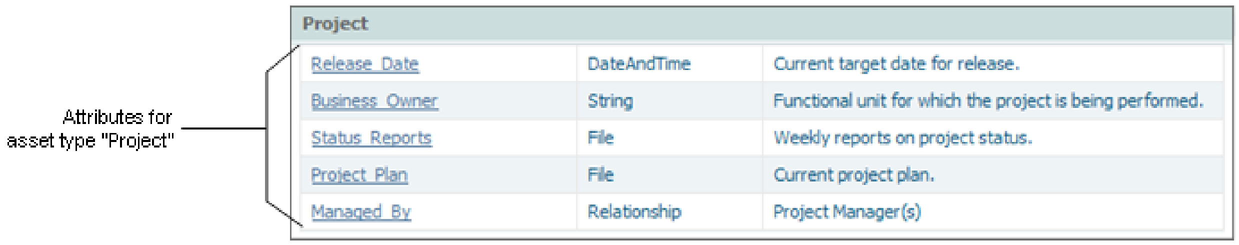
- Add attributes to the type
- Move certain attributes from one profile to another
- Specify which profiles are to be displayed for the type
- Change the type's system-property settings (for example, specify whether the type supports versioning or can be used with design/change-time policies)

Creating Custom Asset Types

Besides customizing the predefined asset types that are installed with CentraSite, you can also define custom types of your own. For example, if you wanted to include items such as service requests, IT projects and source code libraries in your registry, you would create a custom type for each of these entities.

Note: Before creating a custom type, always check to see whether CentraSite provides a predefined type that you might be able to customize and use. Customizing one of CentraSite's predefined types will save you time, especially if the type requires a file importer.

Before creating a custom type, you must first decide which aspects of an entity you want to model in the registry. If you were creating a type to represent IT projects, for example, you might want to capture characteristics such as the name of the project requestor, the lines of business the project is expected to affect, the project plan, the project manager and the project's expected completion date. After you decide which specific characteristics and qualities you want to model, you can create a custom type that includes a corresponding *attribute* for each of those characteristics or qualities.



Project		
Release Date	DateAndTime	Current target date for release.
Business Owner	String	Functional unit for which the project is being performed.
Status Reports	File	Weekly reports on project status.
Project Plan	File	Current project plan.
Managed By	Relationship	Project Manager(s)

Assigning Attributes to a Type

An attribute holds data about an asset. All asset types include a basic set of attributes for general information such as the asset's name, description, creation date and owner. You define additional attributes to hold data that is specific to the type of asset that you want to store in the registry.

When you define an attribute, you specify:

- The type of data that the attribute will hold (e.g., String, Number, Boolean)
- Whether the attribute will hold a single value or multiple values (i.e., an array)
- Whether an attribute is required or optional
- Whether the attribute is read-only

Besides basic data types such as String, Number and Boolean, CentraSite supports the following special types:

Attribute Type	Description
Classification	This type enables users to classify an asset according to a specified taxonomy.
Relationship	This type enables users to establish an association between an asset and another object in the registry.
File	This type enables users to link an asset to a file that resides in CentraSite's repository or exists at a URL-addressable location on the network.

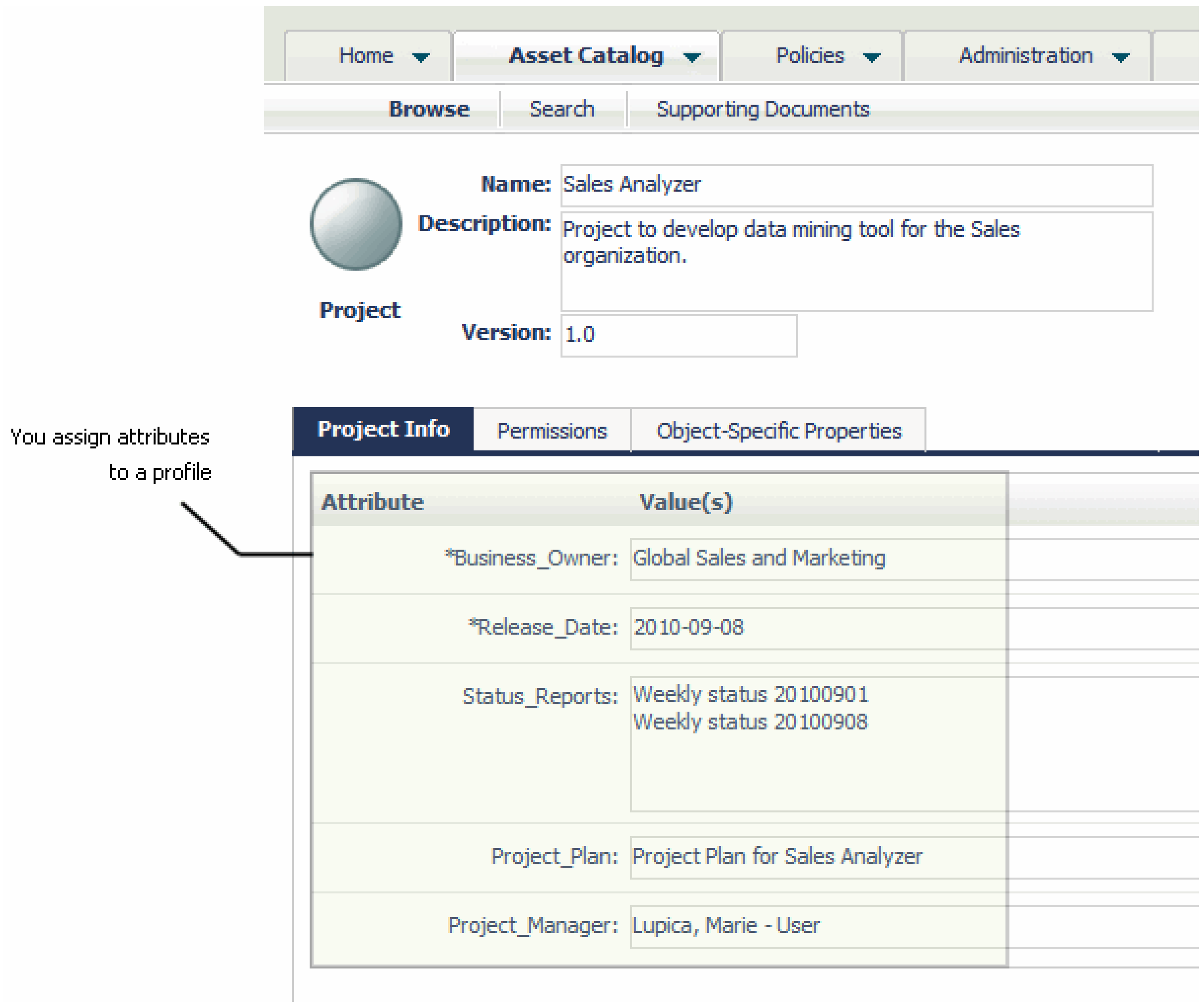
The inclusion of these attribute types facilitate many of the advanced features in CentraSite. It is a good idea to make use of them whenever possible.

For example, let's say you were creating the Project asset type described earlier in this section. Rather than using a String attribute to identify the project manager in this type, you could use a Relationship attribute instead. A Relationship attribute will not only identify the individual who is serving as the project manager, it will also provide the additional benefits of 1) enabling users to obtain detailed information about the project manager (because the attribute itself will link users to the actual User object for that individual), and 2) allowing the relationship to be discovered and reported by CentraSite's Impact Analysis feature. For example, one could use the Impact Analysis feature to locate all of the projects managed by a particular individual.

Assigning Attributes to Profiles

A profile defines a collection of attributes that are meant to be grouped together for presentation purposes.

The attributes associated with a profile are displayed on individual tabs in CentraSite Control



Name: Sales Analyzer
Description: Project to develop data mining tool for the Sales organization.
Version: 1.0

Attribute	Value(s)
*Business_Owner:	Global Sales and Marketing
*Release_Date:	2010-09-08
Status_Reports:	Weekly status 20100901 Weekly status 20100908
Project_Plan:	Project Plan for Sales Analyzer
Project_Manager:	Lupica, Marie - User

When you define a new asset type, you specify on which profiles the type's attributes are to be displayed.

All asset types include several generic profiles. Among others, these include:

- *Audit Log profile*—Displays the history of changes to the asset (including changes in an asset's lifecycle state).
- *Consumers profile*—Displays the users and/or applications that are registered consumers of an asset.
- *Permissions profile*—Displays instance-level permissions for an asset.

- *Classifications profile*—Displays an asset's classifiers.
- *Associations profile*—Lists the registry objects to which the asset is related.

The information on the generic profiles is generated by CentraSite. You cannot customize the content of these profiles or add attributes to them. You can, however, select which of these profiles you want CentraSite to include when it displays an asset of a defined type.

To display the attributes that you define for an asset type, you create custom profiles and assign the attributes to them. CentraSite does not require an attribute to be assigned to a profile. However, if you do not assign an attribute to a profile, the attribute will not be visible in the user interface. You can assign an attribute to multiple profiles if you want it to appear on multiple profiles (tabs) in the user interface.

Note: If you want to provide different views of an asset to different users or groups, divide the attributes among profiles in a way that enables you to use profile permissions to selectively show or hide the appropriate set of attributes to different users or groups.

Creating Custom Asset Types that can be Imported from an Input File

The CentraSite Control user interface enables users to add assets to the registry in the following ways:

- Users can create an asset "from scratch", meaning that they manually assign values to the asset's attributes in the CentraSite Control user interface.
- Users can import an asset from an archive file (a file that contains objects that have been exported from an instance of CentraSite).
- Users can import the asset from an input file. To add an asset in this way, CentraSite must be configured with an "importer" that can read the input file and generate an instance of the specified asset type from it.

CentraSite includes importers for the following types of assets:

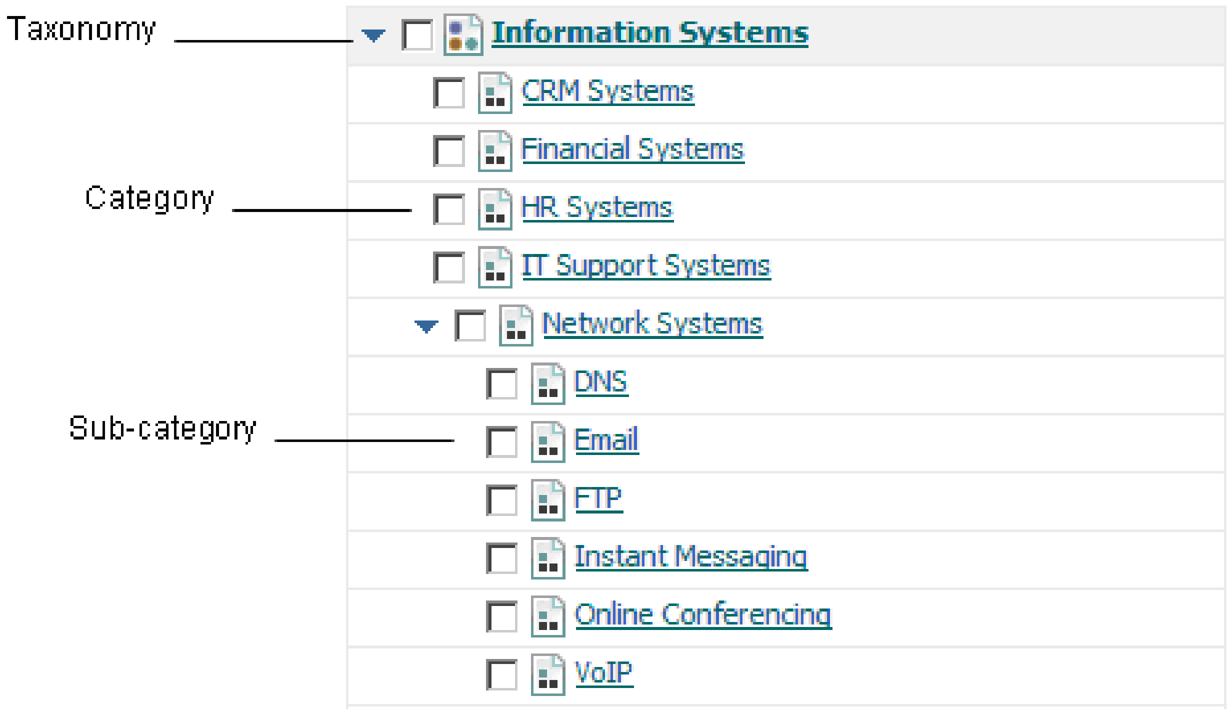
Type of Asset	Required Input File
Web Service	Web Services Description Language (WSDL) file
XML Schema	XML Schema Definition (XSD) file
Business Process	Business Process Execution Language (BPEL) file

If you want your users to be able to generate an instance of a custom asset type from an input file, you must build a custom importer and register it in CentraSite. You can find information about developing a custom importer in the SOALink Cookbook.

Defining and Using Taxonomies

A taxonomy is a hierarchical classification scheme. In CentraSite, you use taxonomies to classify objects in the registry. Taxonomies enable you to filter, group and sort the contents of the registry.

A taxonomy consists of a name and zero or more *categories*. A category represents a classification within the taxonomy. A category can have multiple levels of sub-categories.



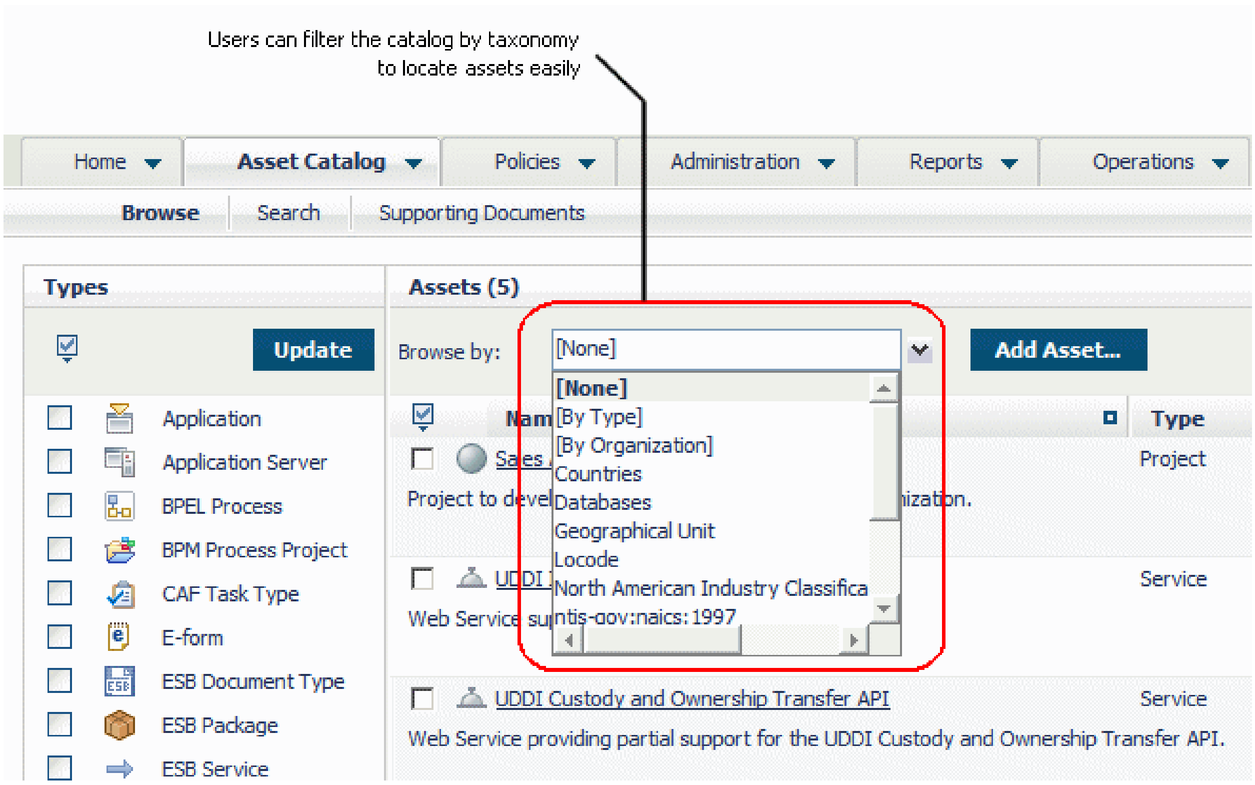
Classifying Assets Using Taxonomies

When users publish assets to CentraSite, they can classify the assets in two ways:

- By directly assigning values to an asset's *Classification attributes*. If an asset's type includes one or more Classification attributes, users can classify the asset by simply setting these attributes.
- By assigning ad-hoc classifiers to an asset's *Classifications profile*. This profile enables users to classify an asset by any available taxonomy defined in CentraSite. It allows users to assign classifiers to an asset in cases where the asset itself does not include any explicit Classification attributes or does not include the needed type of Classification attribute.

How Taxonomies Help Users Locate Assets

Classified assets are easier for users to locate because CentraSite includes convenient tools for filtering, reporting and querying the registry by taxonomy. For example, the Browse page in CentraSite Control enables users to browse the asset catalog according to a specified taxonomy. Additionally, the advanced search feature in CentraSite Control enables users to query the registry for assets that are classified a particular way. By classifying assets, you enable users to discover them using these tools.



Sales ...
Project to devel... ...ization.

UDDI ...
Web Service su...

UDDI Custody and Ownership Transfer API
Web Service providing partial support for the UDDI Custody and Ownership Transfer API.

Note: If you want users to be able to browse the asset catalog by a taxonomy, you must enable the taxonomy's "Taxonomy is browsable" property. If this property is not enabled, it will not appear in the catalog browser's **Browse by** drop-down list.

Using Taxonomies to Target the Execution of Design/Change-Time Policies

Design/change-time policies execute when events within the policy's scope occur in the registry. The scope of a policy specifies to which type of registry objects the policy applies (e.g., Service objects, Policy objects, User objects) and during which types of events the policy is triggered (e.g., a PreCreate event, a PostCreate event, a PreStateChange event).

Classifying assets can help you create highly targeted design/change-time policies, because the scope of a policy can be additionally constrained to objects that are classified in a specified way. For example, instead of applying a particular policy to all Application Server assets, you might want to restrict the policy to just the Application Server assets that are classified by the "APAC" category from the "Domains" taxonomy.

When you define a custom asset type, think about whether you will need to apply different design/change-time policies to specific subsets of that type. If so, make sure the asset type includes a Classification attribute that can be used to distinguish those subsets. (Consider making this a required attribute to ensure that users do not forget to classify assets of this type.)

The Scope of a Taxonomy

Like types, taxonomies are system-wide objects, meaning that they apply to all organizations (i.e., all organizations have access to the same global set of taxonomies). You cannot restrict a taxonomy to a specific organization.

Taxonomies are also visible to all users. You can give specific users Modify or Full instance-level permissions on taxonomies, but you cannot revoke a user's View permission. All users (including guest users) can view the taxonomies defined within an instance of CentraSite.

The Predefined Taxonomies Installed with CentraSite

CentraSite installs a number of standard taxonomies that you can use to classify assets.

These include:

- ISO 3166 Country Codes
- North American Industry Classification System 2002 (NAICS)
- ThomasNet Supplier Registry
- Product and Service Category System: United Nations Standard Products and Services Code (UNSPSC)

CentraSite also includes a number of special-purpose taxonomies that it uses for its own internal classification of registry objects.

You cannot delete any of the predefined taxonomies installed with CentraSite or modify their category structure. You can, however, modify certain attributes and properties for these taxonomies. Additionally, you can suppress them in the user interface. For example, if your users will never use the NAICS taxonomies that CentraSite provides, you can remove these taxonomies from the user interface.

Defining Custom Taxonomies

In addition to using the taxonomies that CentraSite provides, you can create your own custom taxonomies.

When you include a Classification attribute in a type, you usually need to create a corresponding taxonomy for the attribute (unless the required taxonomy already exists in the CentraSite registry). For example, let's say you decide that you want to classify your Application Server assets according to the domain in which they reside. To do this you would first create a custom taxonomy that identifies the various domains in your environment. Then, after the taxonomy exists, you would customize the Application Server asset type and add a Classification attribute to it that enables users to classify application server assets by the "Domain" taxonomy.

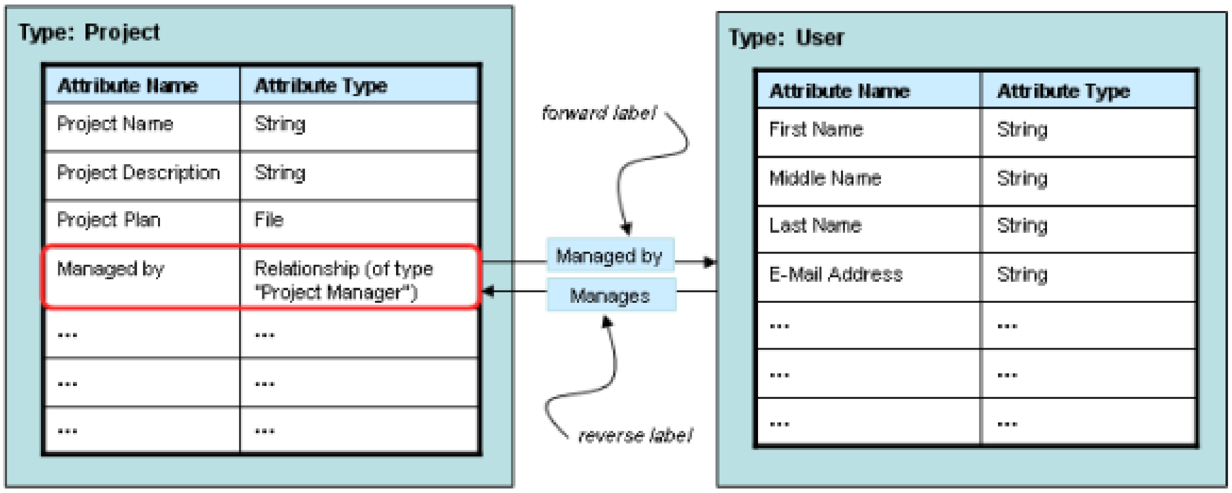
Creating Custom Association Types

An association type describes a type of relationship that can exist between objects in the registry.

An association type has a name, a forward label (which describes the relationship of the source object to a target object) and an optional reverse label (which describes the relationship of the target object to the source object).

You use association types to define Relationship attributes in an asset type. In the following example, a Relationship attribute called "Managed By" has been included in the Project asset type to associate a project asset with the user that manages the project.

You Use Association Types to Define Relationship Attributes in Object Types



Type: Project

Attribute Name	Attribute Type
Project Name	String
Project Description	String
Project Plan	File
Managed by	Relationship (of type "Project Manager")
...	...
...	...
...	...

Type: User

Attribute Name	Attribute Type
First Name	String
Middle Name	String
Last Name	String
E-Mail Address	String
...	...
...	...
...	...

How Association Types Are Used to Relate Assets to Other Objects

When users publish assets to the registry, there are two ways in which they can relate an asset with other objects in the registry.

- *By establishing the relationship using an asset's Relationship attributes*. If an asset's type includes one or more Relationship attributes, users can relate an asset to other objects in the registry by simply setting these attributes.
- *By establishing an ad-hoc association using the asset's Associations profile*. If an asset's type includes the Associations profile, users can relate assets of that type with other objects on an "ad hoc" basis. Using this profile, users can relate an asset to virtually any other object in the registry (assuming they have view permission on the target object).

How Association Types and Relationship Attributes Support Impact Analysis

CentraSite's Impact Analysis feature reports the associations that exist among objects in the registry. By viewing an Impact Analysis report for an asset (either in graphical or tabular form), users can quickly determine to which objects the asset is related.

When you include Relationship attributes in an asset type, you not only enable users to specify the objects to which an asset is related, you enable the relationships to be discovered and reported by the Impact Analysis feature.

Creating Custom Association Types

CentraSite provides numerous predefined association types for you to use to create Relationship attributes. However, you can also create custom association types as needed.

Like types and taxonomies, association types are system-wide objects. They apply to all organizations defined in the registry (i.e., all organizations within an instance of CentraSite have access to the same global set of association types). You cannot restrict the use of an association type to a specific organization.

Working with Asset Types, Taxonomies and Association Types in a Multi-Stage Environment

If you are working in a multi-stage environment, it is important to "master" your custom asset types, taxonomies and association types on one stage and then promote them to the other stages. *Do not* attempt to manually define these objects in each stage. Doing this will create objects that are equivalent, but not identical. That is, the objects will have the same attributes, but they will not have the same Universally Unique Identifier (UUID). It is the UUID that uniquely distinguishes an object in the registry.

When objects are imported into CentraSite (either through an import process or a promotion process), CentraSite uses the UUID to determine whether an object that you are importing already exists on the target instance of CentraSite. If the UUID does not already exist, CentraSite adds the imported object to the registry. If an object with the same UUID exists in the target registry, and the object that you are importing has a more recent timestamp than it, CentraSite automatically replaces the object in the target registry with the one that you are importing.

Important: Because the import process uses timestamps to determine whether an object in an archive file is more recent than the one that exists in the target registry, it is important that the system clocks on all of the participating stages are synchronized.

When you promote an asset from one stage to another, CentraSite also promotes the asset's type and the taxonomies that it uses. If you have manually defined these objects on the target instance of CentraSite, they will be duplicated, instead of replaced, during the promotion process. This will create a confusing situation wherein you have two instances of the same asset type and/or taxonomy on the target instance of CentraSite.

To avoid this condition, always create your custom asset types, taxonomies, and association types on the first stage of a multi-stage deployment and export those objects to the registries that host the subsequent stages of the lifecycle.

Note: Association types that the asset uses are not automatically exported with an asset. If the asset uses custom association types, you must export the association types separately and import them on the other stage(s) before you import the asset itself.

Issues to Consider when Customizing Your Registry

CentraSite provides many ways for you to customize the asset catalog. After you install CentraSite, you should customize the predefined asset types provided by CentraSite (if necessary) and create new types, taxonomies and association types as required for your site.

Note: Although you should do the initial customization after CentraSite is installed, you can always add additional asset types, taxonomies and association types as you develop the need for them.

When customizing the catalog for your site, keep the following points in mind:

- You can customize any of the predefined asset types installed with CentraSite by adding attributes to them and/or modifying the content and organization of the profiles associated with the type.
- If you have an asset that is not represented by one of the predefined types provided by CentraSite, you must create a custom asset type for it. If you want users to be able to generate the asset type from an input file, you must also create a custom importer for that type and register the importer in CentraSite.

- Consider using Classification attributes and Relationship attributes instead of ordinary String attributes whenever possible. Among other benefits, these attribute types enable users to more easily discover assets and understand the relationships that an asset has with other objects in the registry.
- In general, use a Classification attribute or an enumerated String instead of an ordinary String attribute when you want the attribute to be more strongly typed.
- Instead of defining multiple asset types to represent variants of the same basic type, consider creating one basic type and using a classification attribute to differentiate them. For example, instead of creating separate asset types for different kinds of Web services (e.g., business services, technical services, security services), use the one basic Web service asset type and use a Classification attribute to classify its variations.
- When you are designing a new asset type, think about the design/change-time polices that you might want to apply to assets of that type. If you need to apply different policies to different sub-sets of the asset type, use a Classification attribute to differentiate the sub-sets.
- If you do not want users to be able to assign ad hoc classifiers and/or associations to instances of a particular type of asset, omit the Classifications and/or Associations profiles from that asset's type.
- If a taxonomy is designed to be used with specific types of assets, specify those types in the taxonomy's **Applicable to Object Types** tab. This will prevent users from using the taxonomy to classify objects with which the taxonomy was not intended to be used.
- You can define taxonomies with multiple levels of sub-categories to create very fine-grain levels of classification. When you do this, users can search for assets that are classified by a specific category *or any of its sub-categories*. For example, the “Service Type” taxonomy shown in the figure below would enable users to locate a specific type of technical service (e.g. a Utility or a Network service) or all "technical services" (i.e., all services that are classified by the Technical Services category or by any of its sub-categories).
- If you are working in a multi-stage environment, always master your custom asset types, taxonomies and association types on one stage and export them to the other stages. *Do not attempt to define these objects manually on each stage.*

9 Defining Lifecycle Models

A lifecycle model describes the distinct steps through which a particular type of SOA asset passes on its way from conception to retirement. In a very simple sense, a lifecycle model enables you to classify assets according to the state they have reached in their lifecycle. It also provides the basis for CentraSite's lifecycle governance capabilities. Using these capabilities, you can steer assets through the different steps of their lifecycle and apply governance controls at significant junctures of the lifecycle process.

How Lifecycle Models Help You Organize Your Assets

One benefit of using lifecycle models is that they give you increased visibility into the development and/or operational status of the assets in your catalog.

When you apply a lifecycle model to an asset type, the lifecycle model itself is treated as a taxonomy by many of CentraSite's browse and search tools. For example, using the catalog browser in CentraSite Control, you can view the contents of your catalog according to a specified lifecycle model. This view enables you to quickly ascertain which assets are in a particular phase of their development lifecycle.

CentraSite's search tools and reporting features also allow you to query assets by lifecycle state. You can use the advanced search feature, for example, to filter assets by their lifecycle state. You can also use CentraSite's reporting facility to examine the lifecycle status of the assets in your catalog.

How Lifecycle Models Help You Govern Your Assets

Lifecycle models enable you to govern your assets more effectively by allowing you to enforce governance controls at various points in an asset's lifecycle.

When you associate a lifecycle model with an asset type, you make it possible to impose design/change-time policies at each step of the asset's lifecycle. These policies enable you to control the transition of an asset from one step of its lifecycle to another by triggering review and approval processes, issuing email notifications, updating permission settings and generally verifying that an asset meets the requirements necessary to enter the next step in its lifecycle.

Note: The following discussion of lifecycle models describes how you can use lifecycle models with assets. CentraSite also uses lifecycle models to manage policies and lifecycle models.

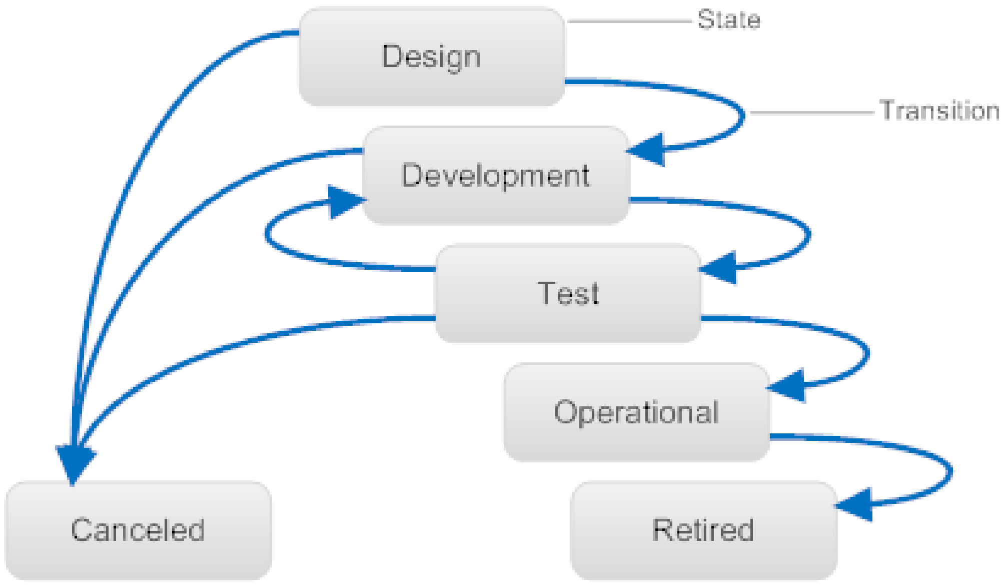
Lifecycle Model States and Transitions

A lifecycle model is composed of *states* and *transitions*.

A *state* represents a distinct step through which an asset passes on its way from conception to retirement. A very simple lifecycle model for an asset might include states such as Design, Development, Test, Operational and Retired. It might also include the Canceled state for assets whose lifecycle is terminated prior to completion.

A *transition* represents the act of switching an asset from one state to another. When you define a lifecycle model, you specify both the states that make up the lifecycle and the transitions that can occur from each state. For example, the Test state might have possible transitions to the Operational, Development and Canceled states. If these are the only three transitions that you define for the Test state, these are the only states to which CentraSite will allow an asset in the Test state to be switched.

Lifecycle models are composed of states and transitions



A lifecycle model must have one initial state. Assets to which the model is applied enter the initial state when they are first added to the registry. Every state except the initial state must have at least one inbound transition associated with it.

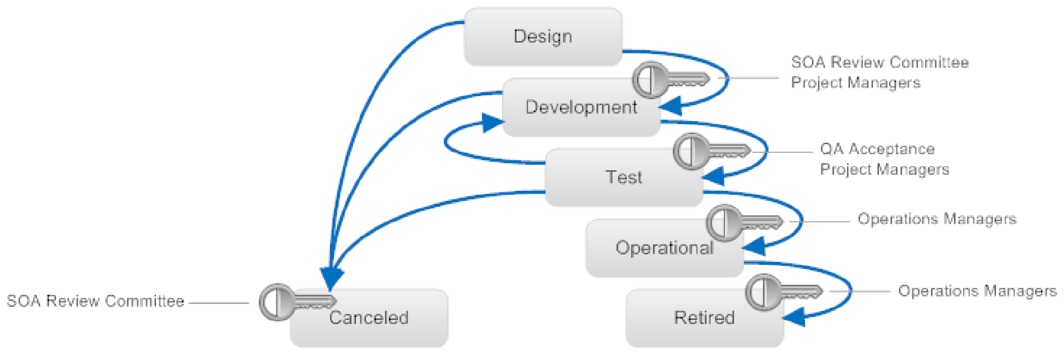
A lifecycle model must have at least one end state. An end state has no outbound transitions associated with it. The lifecycle model depicted in the figure above, for example, has two end states: Canceled and Retired.

Assigning Permissions to Lifecycle Model States

Each state that you define in a lifecycle model includes a set of optional *state permissions*. State permissions enable you to restrict who can transition assets to a specified state. You can assign state permissions to individual users or to groups.

If you do not explicitly assign permissions to a state, any user with Modify permission on an object can switch the object to that state.

You can optionally assign permissions to the states in a lifecycle



When you assign permissions to a state, two sets of users are allowed to switch an asset to that state: 1) the set of users to which you explicitly grant state permission and 2) users who have implicit permission to switch lifecycle states. The set of users who have implicit permission to switch lifecycle states are:

- Users with "Manage System-Wide Lifecycle Models" permission (on objects managed by a system-wide lifecycle model).
- Users with "Manage Lifecycle Models" permission (on objects managed by an organization-specific lifecycle model).
- The owner of the Lifecycle Model.

Note that the group of users with implicit permission to switch states does not include the owner of the asset itself. If you want to give asset owners the ability to switch their assets to a particular state, you must explicitly include them using the state permission settings.

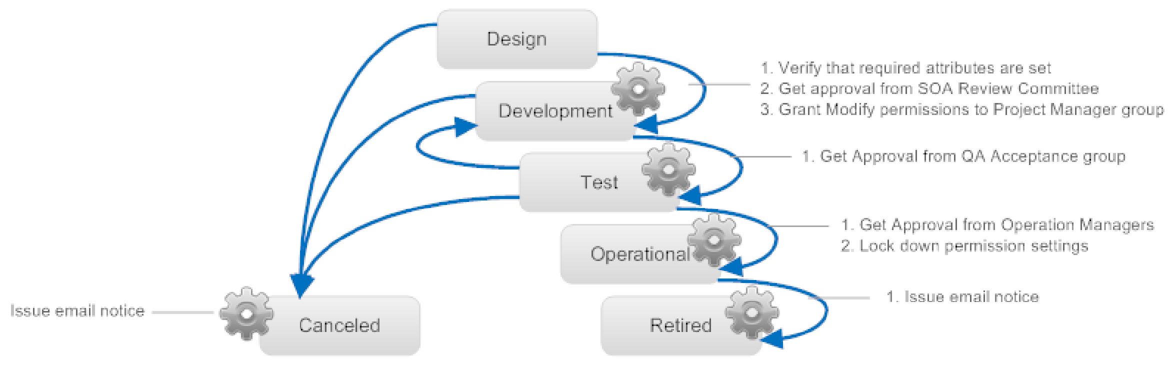
Also note that granting state permission to a user does not, in itself, give the user the ability to switch an asset to that state. The user must also have Modify permission on the asset itself. For example, let's say you give the Users group for organization ABC permission to switch assets to the Development state. Doing this does not mean that any user in organization ABC can switch the assets in organization ABC to the Development state. It means that any user in organization ABC *with Modify permission on an asset* can switch that asset to the Development state.

Note: CentraSite does not allow you to modify a lifecycle model, including its state permissions, after you activate the model (more about activation, below). If you assign state permissions to a lifecycle model, consider assigning the permissions to groups instead of individual users. Doing this will enable you to make simple adjustments to the permission settings by simply modifying the membership of the assigned groups. You will not need to deactivate the model to make these kinds of changes.

Triggering Policies during Lifecycle Model Transitions

You can configure design/change-time policies to execute during the transition points in an asset's lifecycle. For example, you might apply a policy that gives View permission to a specified group of users when an asset enters the Operational state, or you might apply a policy that obtains approvals from a review group before an asset enters the Development state.

CentraSite will trigger polices when specified state transitions occur


Design
Development
Test
Operational
Retired
Canceled
1. Verify that required attributes are set
2. Get approval from SOA Review Committee
3. Grant Modify permissions to Project Manager group
1. Get Approval from QA Acceptance group
1. Get Approval from Operation Managers
2. Lock down permission settings
1. Issue email notice
Issue email notice

When you create a policy that executes on a state change event, you specify whether the policy is to execute immediately before CentraSite actually modifies the asset's state (which is called a *PreStateChange event*) or immediately after CentraSite modifies the asset's state (a *PostStateChange event*).

Generally speaking, you execute policies that perform approval and validation actions on the PreStateChange event. In other words, you use PreStateChange policies to ensure that an asset satisfies the entry criteria for a given state.

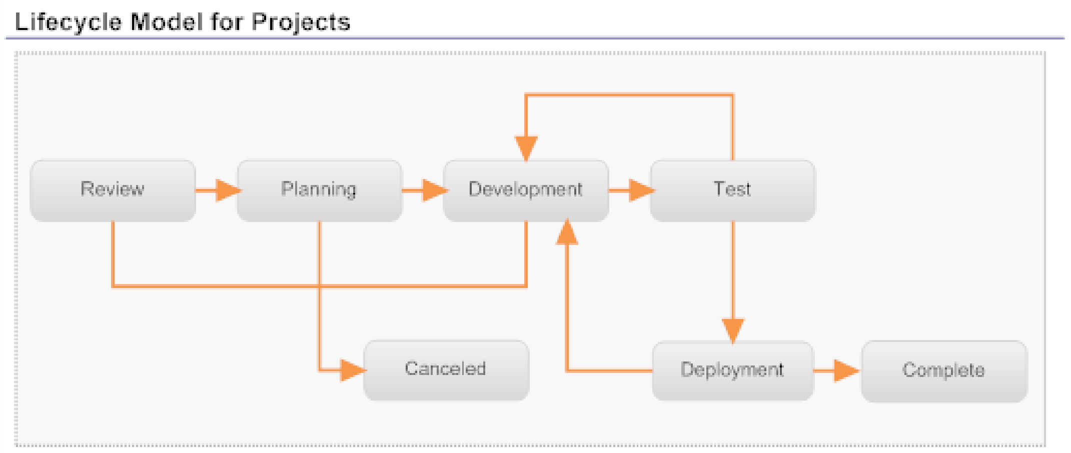
On a PostStateChange event, you typically execute policies that update the asset (e.g., granting instance-level permissions to the users who need to work with the asset in the next phase of its lifecycle) or issue notifications (e.g., sending an email). In short, you use PostStateChange policies to execute actions that are to be carried out only if the asset's state is switched successfully.

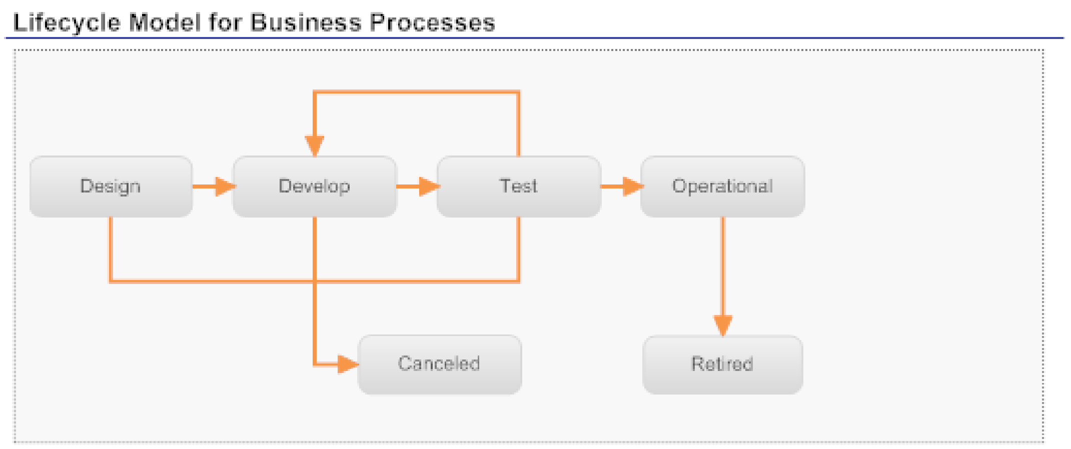
There are, of course, exceptions to the generalizations above. Under some circumstances you might want to set an asset's instance-level permissions or update its attributes in a PreStateChange policy. But generally speaking, you want to perform approval and validation actions in a PreStateChange policy, and you want to issue notifications and perform “state-certain” actions (i.e., actions that should occur only after an object's state has been successfully switched) in a PostStateChange policy.

Associating a Lifecycle Model with an Asset Type

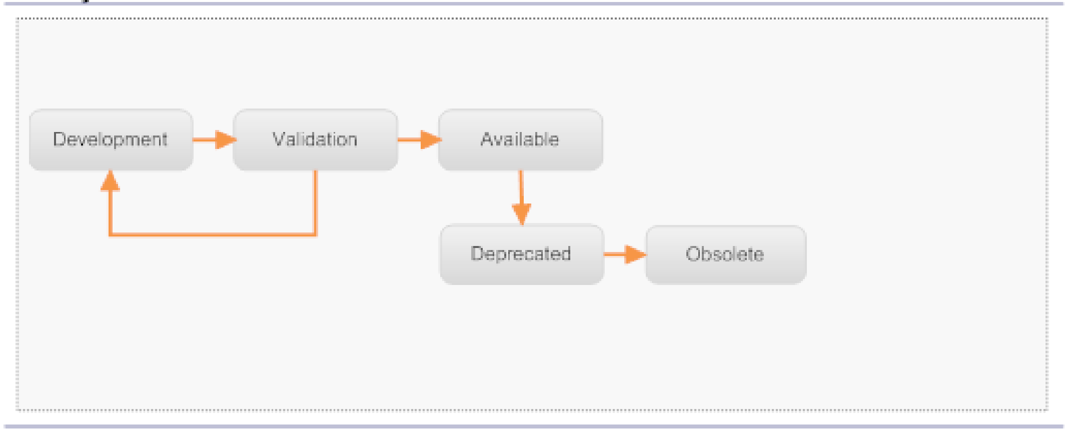
When you define a lifecycle model, you specify the asset types to which the model applies. Because different types of assets usually have different development paths, you generally create models that are specific to a single asset type (i.e., one model for Service assets, one model for XML Schema assets, one model for Business Processes and so forth). However, if multiple asset types have the same lifecycle path, you can apply the same lifecycle model to them all.

You can model different lifecycle models for different assets types





Lifecycle Model for XML Schemas and DTDs



When you apply the same lifecycle model to multiple asset types, you do not necessarily have to apply the same state-change policies to those types. You can trigger different policies depending on the type of asset whose state is changed. If you were using the lifecycle model for XML Schema and DTDs shown in the figure above, you might create one policy that executes when an XML Schema switches to the Available state and another policy that executes when a DTD switches to the Available state.

System-Wide vs. Organization-Specific Lifecycles

A lifecycle model is either *system-wide* or *organization-specific*. System-wide lifecycle models apply to all organizations within an instance of CentraSite. Organization-specific models apply only to a specified organization.

You can use a system-wide lifecycle model when all of the organizations in the registry have to follow the same lifecycle for a particular type of asset. If all your organizations will use the same lifecycle for XML schemas, for example, you could create a system-wide model for the XML Schema asset type. However, if each organization intends to follow a different lifecycle for XML schemas, or if some organizations do not want to impose lifecycle models on their XML schemas at all, you must use lifecycle models that are organization-specific.

Note that when you apply a system-wide lifecycle model to an asset type, all organizations must follow the same path through the lifecycle, however, each organization can define its own set of policies for the lifecycle's transition points. In other words, the state and transition definitions in system-wide model are applied to all organizations, but each organization can associate its own policies with the lifecycle model.

When to Use Lifecycle Models

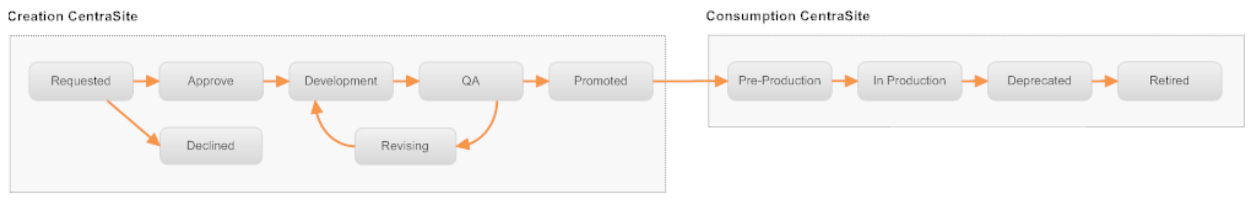
CentraSite does not require you to apply lifecycle models to the assets in your registry. You can create and maintain an asset catalog without them. However, most of CentraSite's policy-based governance controls (e.g., enforcing approval processes, validating attribute settings) can only be applied to assets that have an associated lifecycle model. The use of lifecycle models also makes it easier to manage the assignment of permissions (e.g., granting View permissions to additional organizations) as an asset moves through its lifecycle.

If you want to use design/change-time policies to impose governance controls on a particular type of asset or you want to automate routine management tasks at certain points in the asset's lifecycle (such as setting permission assignments), associate a lifecycle model with the asset type.

Note: For your convenience, CentraSite provides predefined lifecycle models for several types of assets. You can use these lifecycle models “as-is” or customize them to suit your needs. For a list of the predefined lifecycle models, see the section *The Predefined Lifecycle Models Installed with CentraSite* in the section *Customizing Lifecycle Management*.

Lifecycle Stages

Sometimes an asset's overall lifecycle is split across two or more registries. The most common example of this occurs when assets that are in the development and test phases of their lifecycle are maintained in one registry (the “creation” CentraSite) and assets that are deployed (i.e., in production) are maintained in a separate registry (the “consumption” CentraSite).



When a site splits an asset's lifecycle across multiple registries, each participating registry is referred to as a *stage*. Each stage knows about the other participating stages, but does not know the details of the lifecycle that takes place in those stages (that is, the registries that participate in the overall lifecycle are not aware of the specific states and transitions that occur in the other registries).

To model a lifecycle that extends across multiple registries, you must create a separate lifecycle model on each participating registry. Each model describes just the segment of the lifecycle that occurs within its own registry. For example, in the multi-stage lifecycle depicted above, the lifecycle model on the creation registry would consist of the Requested, Declined, Approve, Development, Revising, QA and Promoted states. The lifecycle model on the consumption registry would consist of the Pre-Production, In Production, Deprecated and Retired states.

To indicate that a lifecycle ends on one registry and continues on another, the state that represents the end of an asset's lifecycle on a particular registry will include a pointer to the registry that hosts the next stage of the lifecycle.

Important: Only an end state in a lifecycle model can have a pointer to another stage.

When an asset reaches the end of its lifecycle on one registry, you promote the asset to the next stage of its lifecycle by exporting the asset from the current registry and importing it to the next.

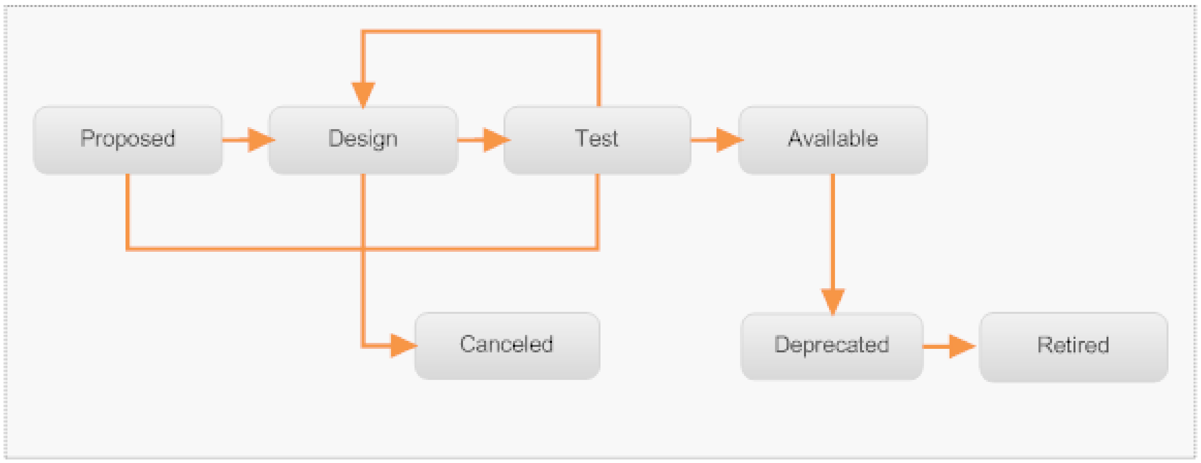
When you import the asset into the registry that hosts the next stage of its lifecycle, CentraSite will first verify that the asset is being imported into the correct stage. It does this by checking the address specified in the “stage” parameter that is included in the archive file with the exported asset. If the address identified in the stage parameter in the archive file matches the registry's own address, CentraSite allows the asset to be imported.

Creating a Lifecycle Model for a Single-Stage Environment

This section describes a simple lifecycle model that you might create for an XML Schema in a single-stage CentraSite deployment. This example assumes that the registry supports both the development (creation) and production (consumption) environments.

In the lifecycle depicted in the following example, the organization that develops an XML schema (the provider organization) retains ownership of that schema throughout the schema's entire life-cycle. As the schema progresses through the lifecycle, the provider organization makes the schema available to others organizations. When the schema reaches the Available state (the point at which it might be used by services that are in production), the Operations organization assumes control over the schema.

A basic lifecycle model for an XML Schema



In general, this lifecycle uses a simple "push it forward" design. With this approach, the user or group responsible for completing a particular development phase switches the schema to the next state when their phase is complete. This action triggers an approval policy, which prevents the state change from occurring unless the "gatekeepers" associated with the next phase approve the change. For example, the developer who proposes the development of a new XML schema pushes the schema to the "Design" state after he or she prepares the specifications necessary for development to begin. However, the schema does not actually enter the "Design" state until the "gatekeepers" of the next phase, in this case the SOA Review group, approve the state change.

The following sections describe each state in more detail. Note that each section includes a description of the state permissions assigned to the state (which identifies those responsible for switching the schema to that particular state) and the policies that are executed during a transition to that state.

The Proposed State

The Proposed state is the initial state in this lifecycle model. An XML schema asset enters the Proposed state when a user adds a schema to the registry. (Typically, the user is a developer, but it might be an analyst or business owner who is requesting the development of the schema.)

The schema remains in the Proposed state while the user gathers the information required for the XML schema to enter the development phase. This might include supplying a formal request document and/or preliminary specifications for the schema. The user might also attach a preliminary XSD file to the schema asset at this point, but that would not be required.

State Permissions. None. An initial state does not have state permissions.

The Design State

The user switches the schema to the Design state after he or she has the documents and specifications needed to obtain approval to develop the schema. Entry into this state must be approved by the SOA Review group.

The schema remains in the Design state until development of the schema is complete and the schema is ready to undergo formal test and validation.

State Permissions. None. Any user with Modify permission on the XML Schema can switch the schema to the Design state.

Policy Execution. When an XML schema is switched to the Design state, a PreStateChange policy executes to obtain approval for the change from the SOA Review group.

The Test State

The developer switches the XML schema to the Test state when the schema is ready for formal testing and validation. Entry into this state must be approved by the QA Acceptance group.

The schema remains in this state while it is in testing. However, a tester might return it to the Design state if the schema fails the testing process and requires additional development or design work.

State Permissions. None. Any user with Modify permission on the XML Schema can switch the schema to the Test state.

Policy Execution. When an XML schema is switched to the Test state, a PreStateChange policy executes to obtain approval for the change from the QA Acceptance group.

A PostStateChange policy executes to give Full permission on the schema to the QA Acceptance group.

The Available State

A member of the QA Acceptance group switches the XML Schema to the Available state when testing is complete and the schema is considered to be ready for use. Entry into this state must be approved by the Operations Acceptance group.

When the schema enters this state, a group within the Operations organization generally assumes control of the schema and is given Full permission on the schema. As necessary, they grant View permission to other users.

State Permissions. The QA Acceptance group is given permission to switch an XML schema asset to this state.

Policy Execution. When an XML schema is switched to the Available state, a PreStateChange policy executes to obtain approval for the state change from the Operations Acceptance group.

A PostStateChange policy executes to give Full permission on the XML schema to the Operations Acceptance group. It removes all other instance-level permissions assigned to the schema.

The Canceled State

A member of the SOA Review group switches the schema to the Canceled state if development of the schema is halted and will not be resumed. A schema might enter this state, for example, if the proposal submitted by the developer is rejected by the SOA Review group because the requested schema already exists in the catalog.

State Permissions. The SOA Review group is the only group that is allowed to switch an XML schema asset to this state.

Policy Execution. A PostStateChange policy sends an email to notify the entire SOA Review group that development of the schema has been terminated.

The Deprecated State

A member of the Operations Acceptance group switches an XML schema to the Deprecated state to indicate that the XML schema is obsolete.

State Permissions. The Operations Acceptance group is given permission to switch an XML schema asset to this state.

Policy Execution. A PostStateChange policy sends an email to notify developers that the XML schema has been deprecated.

The Retired State

A member of the Operations Acceptance group switches an XML schema to the Retired state when the XML schema is no longer available for use.

State Permissions. The Operations Acceptance group is given permission to switch an XML schema asset to this state.

Issues to Consider When Using Lifecycle Models in a Single-Stage Environment

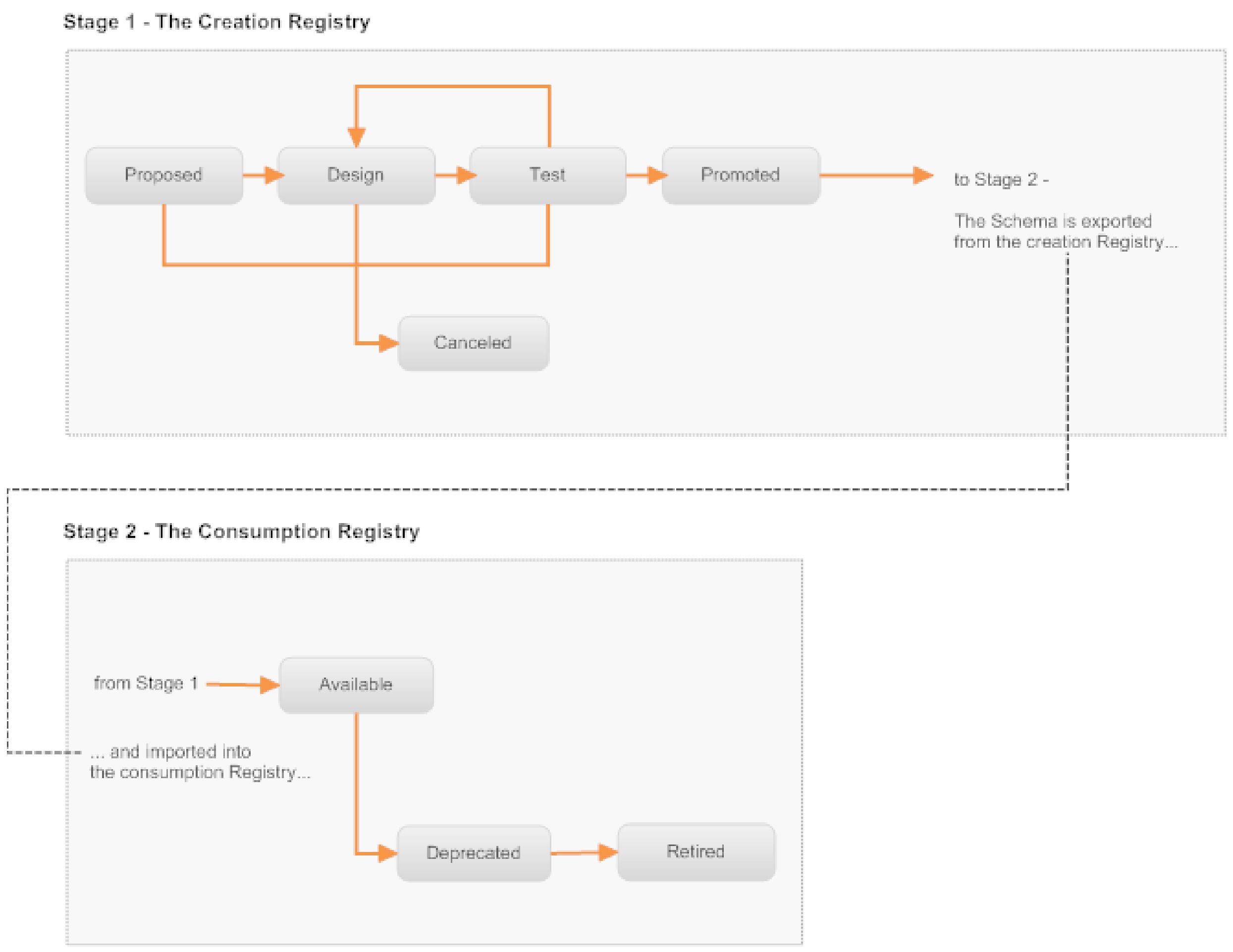
When creating your own lifecycle models for a single-stage environment, keep the following points in mind:

- Be aware that in a single-stage lifecycle design like the one described above, the user who enters the XML schema into the catalog retains ownership of the schema throughout its entire lifecycle. Consequently, when the XML schema reaches the Available state in this lifecycle, the original owner (and any other user with "Manage Assets" permission within the owner's organization) retains the ability to edit and delete the schema. The schema will not be under the sole control of the Operations organization. To address this issue, you might want to move the asset into the Operations organization when it reaches the Available state.
- In any lifecycle design that uses approval polices, it is important to ensure that the members of the approval groups have View permission on the assets that they will be asked to approve. Without View permission on the asset, an approver cannot process an approval request on that asset. To ensure that the appropriate permissions are in effect when an asset reaches an approver, consider including an action in the approval policy that grants View permission to the approval group just before the policy executes the approval action.
- Generally speaking, to change an asset to a particular state, a user must have state permission on that state *and* have Modify permission on the asset itself. However, if the state change triggers a policy that sets instance-level permissions on the asset, the user who triggers the policy *must have Full permission on the asset.* Otherwise the policy will fail. Thus, if your lifecycle design uses policies to set instance-level permissions on an asset during certain state changes, make sure that you give the users who will make those state changes Full permission on the asset.

Creating a Lifecycle Model for a Two-Stage Environment

This section shows how you might take the simple lifecycle for an XML schema described above and modify it for a two-stage deployment. As shown in the figure below, the overall lifecycle of the schema, from the Proposed state to the Retired state, now occurs over two registries.

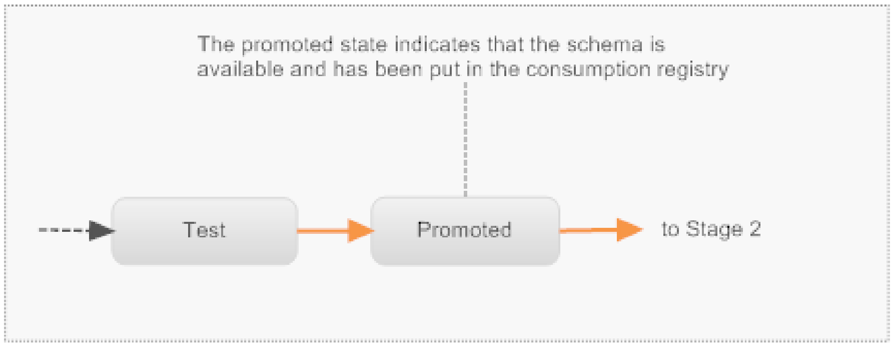
Example Lifecycle for XML Schema Split Over Two Registries


Stage 1 - The Creation Registry
Proposed
Design
Test
Promoted
to Stage 2 -
The Schema is exported from the creation Registry...
Canceled
Stage 2 - The Consumption Registry
from Stage 1
Available
... and imported into the consumption Registry...
Deprecated
Retired

Modeling this kind of lifecycle requires two lifecycle models: one on the creation registry and one on the consumption registry.

The Lifecycle Model on the Creation Registry

The lifecycle model on the creation registry encompasses the development and testing phases of the lifecycle. This lifecycle model includes the Proposed, Design and Test states, just like the single-stage example. The design of these steps in the lifecycle, in terms of state permissions and policy execution, is also the same as the one used by the single-stage example. However, instead of moving directly to the Available state when testing is complete, a schema moves to the Promoted state.


The promoted state indicates that the schema is available and has been put in the consumption registry
Test
Promoted
to Stage 2

The Promoted State

The Promoted state is an end state in this lifecycle model. The **Stages** property is set on the Promoted state. This property indicates that the schema's lifecycle continues on another registry and it identifies the address of that registry.

A member of the QA Acceptance group switches the XML Schema to the Promoted state when testing is complete and the schema is considered to be ready for use. Entry into this state must be approved by the Operations Acceptance group.

After the schema enters the Promoted state, an administrator exports the schema to an archive file. This file is subsequently imported by an administrator on the consumption registry.

State Permissions. The QA Acceptance group is given permission to switch an XML schema asset to this state.

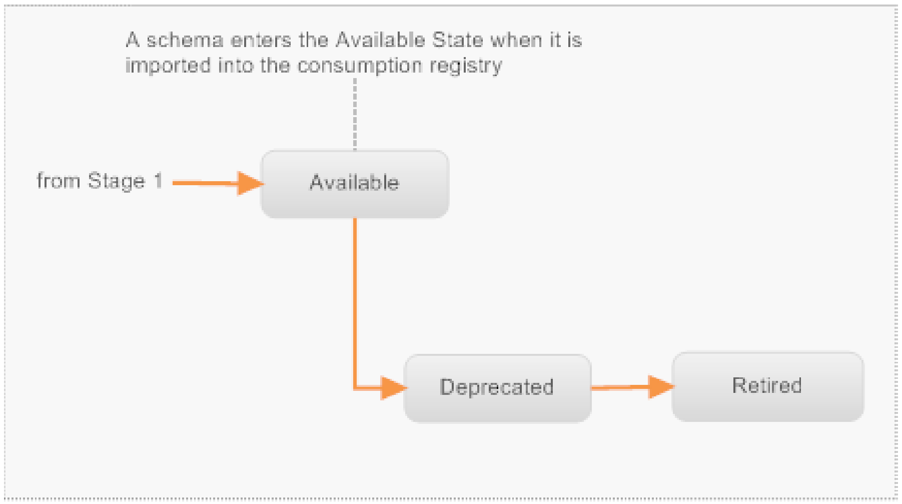
Policy Execution. When an XML schema is switched to the Available state, a PreStateChange policy executes to obtain approval for the state change from the Operations Acceptance group.

A PostStateChange policy executes to give Full permission on the XML schema to a small group of administrators. It removes all other instance-level permissions assigned to the schema.

The Lifecycle Model on the Consumption Registry

The lifecycle model on the consumption registry defines three states: Available, Deprecated and Retired. The Available state is the initial state. A schema will enter this state when it is imported into the registry.

In terms of state permissions and policy execution, the design of this lifecycle model is similar to the last three states of the single-stage example.


A schema enters the Available State when it is imported into the consumption registry
from Stage 1
Available
Deprecated
Retired

Typically, on a consumption registry, an "operations" organization owns and controls all assets in the registry. Users in this organization are the only ones permitted to create assets. (The schema would be imported by one of these users, for example.) All other users are simply consumers. The operations organization grants these users View permission on the assets as necessary.

Issues to Consider When Using Lifecycle Models in a Two-Stage Environment

When creating lifecycle models for a two-stage environment like the examples shown above, keep the following points in mind:

- The user who imports an asset into the consumption registry specifies the organization into which asset will be imported. This user can import the asset to any organization for which he or she has "Create Assets" permission. If the archive file includes system-wide objects, such as asset types or taxonomies, the user who performs the import must also have permission to create those types of objects. Because of the permission requirements that an import process might require, an administrator in the CentraSite Administrator role often performs the import task. (Bear in mind, that whoever imports the objects on the target registry becomes the owner of those objects. For this reason, you might want to have more than one user in a role with the permissions necessary to import all types of registry objects.)
- Be aware that when an asset reaches an end state on a particular stage, the asset remains in that end state on that stage regardless of the state changes that occur in the next stage. For example, when a schema reaches the Promotion state in the example above, the schema's state will no longer change on the creation registry. The remaining states in the schema's overall lifecycle (i.e., the Available, Deprecated, Retired states), are reflected only in the consumption registry. To determine whether a Promoted schema has been moved to the Deprecated or Retired states, a user must view the asset in the consumption registry.

Updating Assets That Are Under Lifecycle Management

If you need to update an asset that has reached the “production” phase of its lifecycle, you have a couple of choices. If you need to make a minor change, for example, you need to correct an attribute setting, add a classifier to the asset or modify the asset's description, an authorized user can simply make the change directly to the production version of the asset. If you are working in a multi-stage environment, you will need to manually apply the updates to the asset in each of the participating registries.

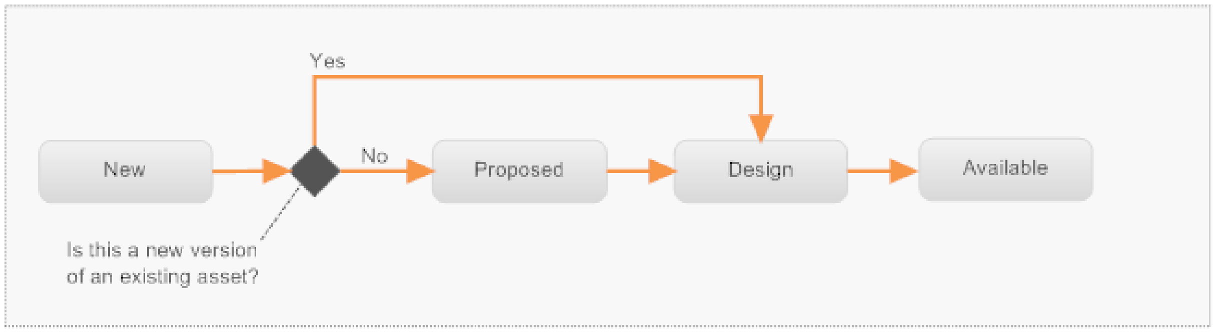
If the changes are substantive, in particular, if they involve changes to the structure of a schema or the definition of an interface, then you should create a new version of the asset. When you create a new version of an asset, the new version enters the initial lifecycle state, just as though it were a completely new asset. The new version of the asset will pass through the entire lifecycle just like any other new asset of its type.

If you are working in a multi-stage environment, you must create the new version on the registry that hosts the first stage of the lifecycle (i.e., on the creation CentraSite). When the new version reaches the end state on that stage, you would promote the new version of the asset to the next stage just as you did with the previous version of the asset.

Creating a Different Lifecycle Path for a New Version of an Asset

For certain asset types, you might want to define separate lifecycle paths for new instances of an asset and new versions of an asset. For example, in a lifecycle for an XML schema like the one shown below, you might want new versions of existing schemas to bypass the Proposed state and go directly to the Design state.

Alternate Lifecycle Path for a New Version of an Asset



Creating an alternate path in a lifecycle requires the use of policies that conditionally change the state of an asset depending on the way in which the asset is classified. In the example shown above, this is achieved by doing the following:

• Defining an initial state (the New state) through which all schemas (new or versioned) pass.

• Creating policies that execute immediately after a schema enters the New state. These policies switch the schema to the “Proposed” state or the “Design” state depending whether the schema is classified as “New” or “Existing”.

To implement a lifecycle like the one above, you must add to the XML Schema asset type a Classification attribute that can be used to classify a schema as either “New” or “Existing”. (You would need to create a custom taxonomy to support this attribute.)

You must also create two policies that execute after a schema enters the New state: one policy that executes when a “New” schema enters the New state (this policy will switch the schema to the Proposed state), and one policy that executes when an “Existing” schema enters the New state (this policy switches the schema to the Design state).

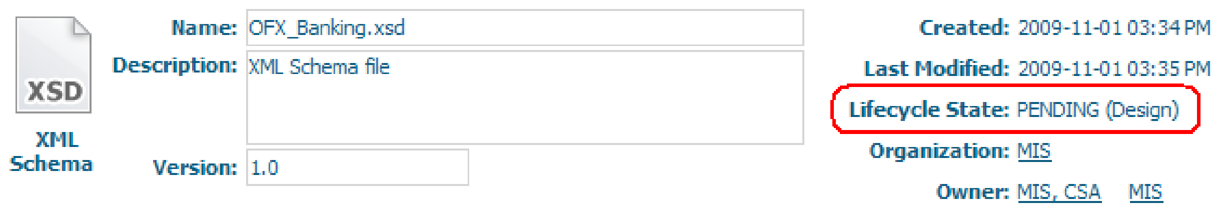
Note: The example above describes how you can use policies to conditionally route an asset between two alternate paths when an asset enters the initial state of its lifecycle. However, you can use this same technique to establish alternate paths at any point in the asset's lifecycle. Its use is not limited to the initial state.

Reverting an Asset to a Previous State

When you switch an asset from one state to another, the asset exists in a “pending” state until the requested state change is complete. While an asset is in the pending state, it cannot be modified.

For most state switches, this is a very brief period of time. However, if the state change involves the execution of policies, it can be quite long (in the case of an approval policy, an asset might remain in the pending state for days).

An object remains in the "pending" state until the requested state change is complete



Name: OFX_Banking.xsd
Description: XML Schema file
Version: 1.0
Created: 2009-11-01 03:34 PM
Last Modified: 2009-11-01 03:35 PM
Lifecycle State: PENDING (Design)
Organization: MIS
Owner: MIS, CSA MIS

An asset can, on occasion, encounter conditions that cause it to become stuck in the pending state. To resolve the situation where an asset becomes stuck in the pending state, a user that belongs to the CentraSite Administrator role can use the **Revert Pending State** command to return the asset to its prior state. After the asset is reverted and the issue that caused the asset to become stuck is corrected, an authorized user can switch the asset to its next lifecycle state again.

Note: Reverting the lifecycle state of an asset does not undo any attribute changes that might have been made by policies that executed during the first state change event. It simply returns the asset's lifecycle property to its previous state. If other attribute changes occurred during the state change event, you will need to undo those changes manually.

Managing Lifecycle Models

This section describes issues relating to the creation and maintenance of lifecycle models.

Activating a Lifecycle Model

Lifecycle models are themselves governed by a predefined lifecycle model. This lifecycle model defines three states: New, Productive and Retired.

When you initially create a lifecycle model it enters the New state. CentraSite does not begin enforcing a new lifecycle model until you *activate* the model by switching it to the Productive state.

After you activate a lifecycle model (i.e., place it in the Productive state), that lifecycle model can no longer be modified. To make changes to the lifecycle model, you must create a new version of the model and make your changes to the new version.

Note: After you retire a lifecycle model, that model cannot be activated again. The Retired state is an end state for lifecycle models.

Versioning or Replacing Lifecycle Models

If you need to make changes to a lifecycle model after the model has been activated (i.e., after you place it in the “Productive” state), you must either create a new version of the existing model or replace the existing model with a completely new model. You cannot modify a lifecycle model directly after it has been activated.

The easiest way to apply changes to a lifecycle model is to generate a new version of the model. This task involves the following basic steps:

1. Creating a new version of the model. During this step, CentraSite creates an exact copy of the existing lifecycle model.
2. Updating the new version of the lifecycle model as necessary (e.g., adjusting its state permissions, inserting additional states, removing states, modifying transition paths, applying it to additional object types).
3. Activating the new version. This step automatically activates the new version and retires the old version.

When you activate a new version of a lifecycle, instances of assets that were created under the old lifecycle model will automatically switch to the new lifecycle model if they are in a state that exists in the new model. Otherwise, they will continue to follow the old lifecycle model until they are switched to a state that exists in both models. At that point, they will switch to the new lifecycle model. For more information about how CentraSite maps the states between old and new versions of a lifecycle model, see the user documentation for lifecycle models.

You can also apply changes to a lifecycle by defining an entirely new lifecycle model. To put the new model into effect, you must retire the existing model and then activate the new model. When you change a lifecycle this way, the objects that were created using the old model will complete their lifecycles under the old model. Objects that are created after the new model is activated will follow the new model.

Modifying the Predefined Lifecycle Model for Lifecycle Models

The predefined lifecycle model that CentraSite uses for lifecycle models is made up of three states: New, Productive and Retired. This lifecycle is generally adequate for most environments. However, you can make certain types of customizations to it if necessary.

For information about the ways in which you can customize this lifecycle model, see the section *Customizing Lifecycle Management > Lifecycle Model for Lifecycle Models* .

10 Defining Design/Change-Time Policies

Design/change-time polices provide governance controls that you can use to effectively administer and manage Web services and other assets within your SOA environment. Design-time policies enable you to control the acceptance of assets into the registry and manage their eventual deployment into the runtime environment. You can use design/change-time policies to ensure that assets entering the SOA environment conform to organizational standards and conventions, meet the architectural requirements of your enterprise and adhere to industry best practices. You can also use policies to execute standard procedures, such as initiating review processes, issuing notifications and granting instance-level permissions at key junctures in an asset's lifecycle.

The following sections generally describe design/change-time policies and explain how you might apply them at your site. For additional information about design/change-time policies, including specific steps for the procedures described in these sections, see the section *Working with Design/Change-Time Policies*.

What is a Design/Change-Time Policy?

A design/change-time policy defines a sequence of user-specified tasks that CentraSite is to execute when designated events occur in the registry. For example, a policy could instruct CentraSite to perform any of the following tasks:

- Verify that a schema name conforms to specified conventions when a new schema is added to the catalog.
- Submit a Web service to a review panel for approval before the service enters the Design state.
- Send an email notification to the IT organization when a service is deployed.
- Change the permission settings on an asset when the asset switches to the Available state.
- Assign a user in a given role to a particular group when a new user is added to CentraSite.
- Alter an asset's lifecycle path depending on the way in which the asset is classified.

A design/change-time policy has two major elements: it has an *action list,* which specifies the tasks (policy actions) that are to be executed, and it has a defined *scope*, which specifies when the action list is to be executed.

Policy Actions

A policy action is a programmed task, written in Java or Groovy (a scripting language). A policy action can perform any type of work you require (e.g., sending an email, validating an attribute setting, submitting an approval request, updating a database). It can have one or more input parameters. An action that sends an email message, for example, will include input parameters that specify the text of the message and to whom it is to be sent. A policy action returns a completion code that indicates whether it completed its task successfully.

The action list in a policy can contain one or more actions. CentraSite executes the actions in the order that they appear in the list. If an action does not complete successfully, CentraSite immediately exits the policy and bypasses any remaining actions in the list. Policy failures are recorded in CentraSite's policy log.

CentraSite is installed with a library of "built-in" policy actions that you can use to construct policies. You can also develop your own custom actions using Java or Groovy.

Policy Scope

The policy's scope specifies the conditions under which CentraSite is to execute the policy. It consists of two main parameters: *Event Type* and *Object Type*.

- *The Event Type parameter* specifies the events to which the policy applies. An event represents a specific point during a registry operation when policies can be executed. Such points include the PreCreate event (the point in time just before CentraSite saves a new instance of an object to the registry), the PostCreate event (the point immediately after CentraSite saves a new instance of an object to the registry) and the PreDelete event (the point in time immediately before CentraSite deletes an object). Other events include the points in time before and after an update operation and before and after a state change. For a complete list of the supported event types, see the topic *Events during Which Design/Change-Time Policies Can Be Enforced* in the section *Working with Design/Change-Time Policies > Functional Scope* .
- *The Object Type parameter* specifies the types of objects to which the policy applies. Policies can be applied to any type of asset and to several other types of registry objects. For a complete list of supported objects, see the topic *Objects on Which Design/Change-Time Policies Can Operate* in the section *Working with Design/Change-Time Policies > Functional Scope* .

Together, the event type and the object type determine when the policy will execute. You can make the scope as narrow or as broad as you need. That is, you can target the policy for one particular type of event and object (e.g., a PreDelete event on an XML Schema) or apply the policy to multiple events and objects (e.g., a PreCreate, PreUpdate or PreDelete event on an XML Schema, a Service or a BPEL Process).

Refining a Policy's Scope with Additional Selection Criteria

You can optionally refine the scope of a policy to narrow the set of objects to which the policy applies. To do this, you include additional selection criteria based on an object's Name, Description or Classification properties. For example, you might create a policy that applies only to Application Servers whose name includes the string “myDomain.com:” or to Application Servers that are classified as Software AG Runtime.

The ability to execute policies based on object classification is an especially effective way to selectively apply policies to objects. As described in ***Creating a Different Lifecycle Path for a New Version of an Asset***, you can use this technique to route assets between alternate paths in a lifecycle model. However, this technique can generally be used any time you want to apply a policy to a particular subset of objects within the specified object types.

Scope of a Policy Action

A policy action also has a declared scope. The action's scope specifies the object and event types with which the action can be used. Some actions have very specific object-type and event-type requirements. For example, you can use the “Validate Policy Deactivation” action only during a PreStateChange on a Policy object. Other actions support a broad range of object and event types.

A policy can contain only actions that support the full set of object types and event types specified by the policy's scope. For example, if you create a policy that executes on the PreCreate and PreUpdate events for XML Schemas, it can only contain actions whose scope includes the PreCreate and PreUpdate event types and the XML Schema object type.

When you create a policy, CentraSite's user interface will only allow you to select actions that satisfy the specified scope of the policy. If you subsequently change the policy's scope, CentraSite will not allow you to save the updated policy unless all of its actions support the policy's new scope.

System-Wide vs. Organization-Specific Policies

A design/change-time policy is either system-wide or organization-specific. System-wide policies apply to all organizations within an instance of CentraSite. Organization-specific models apply only to a specified organization.

Whether a policy is system-wide or organization-specific affects the policy's scope. When a policy is organization-specific, its scope is limited to the set of objects that belong to a specified organization. For example, if you create a policy that executes on a PreDelete event for XML Schemas, and you make that policy specific to organization ABC, that policy will execute only when XML Schemas in organization ABC are deleted. If you make it system-wide, it will execute when an XML Schema in any organization is deleted.

Note: Once you create a policy, its organizational scope is fixed and cannot be changed. That is, if you create a policy whose scope is specific to organization ABC, you cannot change its scope to make it system-wide or switch it to another organization. You must create a new policy and set its organizational scope in the new policy as needed.

Policy Priority

A policy has a priority value, which CentraSite uses when an event triggers multiple policies. You can give a policy a priority value between 11 and 9999 (inclusive). Priority 11 is the default. Values less than 11 and greater than 9999 are reserved for system use.

A policy's priority value is used only when an event triggers multiple policies. When this occurs, CentraSite examines the priorities of the selected policies and executes the policies serially, in priority order, from the lowest value to the highest value (that is, it executes the policy with the *lowest* value first). Each policy in the series is executed to completion before the next one begins.

For example, if an event were to trigger the following policies, CentraSite would execute the policies in the following order: B, A, C (as determined by the priority assignments 11, 25 and 100, respectively).

Policy	Priority
A	25
B	11
C	100

If multiple policies have the same priority, they will be executed serially, but their order is indeterminate. (That is, you cannot predict the order. CentraSite will choose the order at enforcement time.)

Pre-Operation and Post-Operation Event Types

Many of the event types to which you can apply policies represent *pre-operation* events or *post-operation* events.

- Pre-operation events occur immediately before a requested operation is performed on a registry object. They include the following event types: PreCreate, PreUpdate, PreStateChange and PreDelete. When you apply a policy to a pre-operation event, CentraSite executes the requested operation only if the pre-operation policy executes successfully. You generally apply policies at these points to prevent the requested operation from being executed unless an object satisfies the verification checks performed by the policy.

- Post-operation events occur immediately after a requested operation is performed on a registry object. They include the following event types: PostCreate, PostUpdate, PostStateChange, PostDelete. When you apply a policy to a post-operation event, CentraSite executes the policy only if the requested operation is performed successfully. Post-operation polices are often used to notify users (via email) that certain changes have occurred in the registry, to update specified attribute values and to assign permission settings on an object.

What Happens When a Design/Change-Time Policy Executes?

When an event occurs in the registry, CentraSite determines which policies are “within scope” and executes those policies in priority order (from lowest assigned value to the highest assigned value). If an action within a policy fails, CentraSite immediately exits the policy. It does not execute any of the remaining actions in the policy, nor does it execute any remaining policies that are within scope of the event.

If the policy was triggered by a pre-operation event (e.g., a PreCreate event or a PreStateChange event) the requested operation is also not executed. For example, if a user attempts to add an XML Schema to the catalog, and the schema does not satisfy a validation policy that is triggered by the PreCreate event for XML Schemas, CentraSite will reject the user's request to add the new schema to the catalog.

Policy failures are written to CentraSite's policy log. From the Inbox page in CentraSite Control, users can view the failed policies that were logged during the events that they initiated. Administrators with "View Policy Log" permission can view and query the entire log using CentraSite Control's Logging feature.

OnTrigger Policies

CentraSite provides a special event type called an OnTrigger event. Policies that you create for this event type can be run “on demand” from the CentraSite Control user interface. Anyone who has View permission on an OnTrigger policy can execute the policy on demand.

When you run a policy on demand, CentraSite applies the policy directly to each object instance in the registry that:

- Is of a type specified in the policy's object scope.
- Satisfies all conditional criteria specified by the policy (i.e., Name, Description and/or Classification criteria that the policy specifies).
- Is an object on which the user running the policy has View permission. If the policy is organization-specific, the policy is applied to only the objects that satisfy the preceding criteria and

belong to the organization specified by the policy. If the policy is system-wide, the policy is applied to all objects in the registry that satisfy the preceding criteria.

Administrators often use OnTrigger policies to assign permissions to a specified set of objects instead of manually setting permissions on individual objects using the user interface.

Typical Uses for Design/Change-Time Policies

The following sections describe typical uses for design/change-time polices.

Using Policies to Initiate Reviews and Approvals

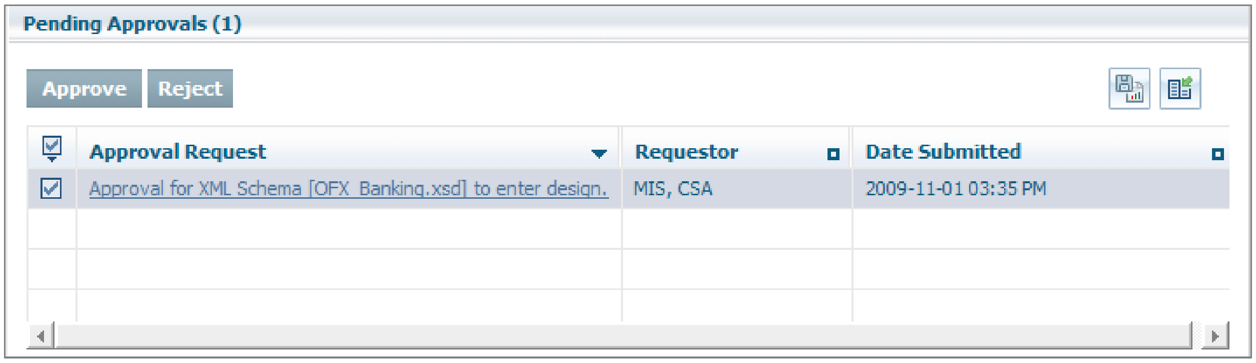
Enforcing a review and approval process is a common use of a design/change-time policy. To create this type of policy, you include one of CentraSite's approval actions in the policy. An approval action identifies the group of users (the approver group) whose approval is required in order to complete the policy successfully. When you configure an approval policy, you can specify whether approval is required from just one approver in the group or from all approvers in the group.

You apply approval polices to PreStateChange events. Thus, to use approval policies on assets, those assets must be under lifecycle management.

Note: Approval policies can also be used with an OnConsumerRegistration event. This use case is described later in this section.

When CentraSite executes an approval policy, it initiates an approval workflow. Users who are designated approvers review the request on the **Pending Approvals** tab of their Inbox page in CentraSite Control user interface. If the approvers approve the request, CentraSite executes the requested state change. If an approver rejects the request, the policy fails and the requested state change is not executed.

Pending approvals appear in the approvers inbox in CentraSite Control



✓	Approval Request	Requestor	Date Submitted
✓	Approval for XML Schema [OFX_Banking.xsd] to enter design.	MIS, CSA	2009-11-01 03:35 PM

Users who submit requests that require approval can view the status of their requests on the **Approval Requests** tab of their Inbox page in the CentraSite Control user interface.

Using Policies to Validate Assets

Validation is another common task that you can perform using design/change-time policies. You use validation policies to ensure that assets conform to organizational norms and standards before they are accepted into the catalog or enter a critical lifecycle state. For example, you might create validation policies to perform the following types of tasks:

- Ensure that a service satisfies certain naming conventions, that it exists within a specified namespace and/or supports specified protocols.
- Ensure that an asset has been classified by a specified taxonomy and/or includes required attribute settings.
- Ensure that a service complies with Web Service Interoperability (WS-I) standards (Basic Profile 1).
- Prevent an asset from being deleted unless it is has reached a specified state within its lifecycle.

CentraSite provides built-in actions that you can use to perform many validation tasks. You can also create custom actions to perform validation tasks that are specific to your environment.

Using Policies to Modify Assets

You can use design/change-time policies to make changes to assets when certain events occur. Setting instance-level permissions on an asset is an example of an update you might make using a policy. Other kinds of changes you might make include classifying an asset, setting a specified attribute when an asset completes a series of tests and changing an asset's lifecycle state.

CentraSite provides built-in actions that you can use to perform many kinds of updates to assets. You can also create custom actions to modify assets.

Using Policies to Issue Notifications or Update External Systems

Notifying individuals when certain events occur in the registry is another common use case for design/change time policies. CentraSite includes a built-in email action that you can use for this purpose.

You might, for example, use this action to send an email to key users in your IT organization when a service switches to the Production state of its lifecycle. You might also create custom actions that would send messages to and/or trigger processes on external systems when certain events occur in the CentraSite registry.

Using Policies to Execute a Consumer Registration

If you want to use CentraSite's consumer-registration feature, you must first create a consumer-registration policy. CentraSite will not enable the consumer-registration feature until you create this policy.

The consumer-registration feature in CentraSite enables users to register users and/or applications as consumers of an asset. To complete the registration process, the owner of the asset must first review and accept the user's request. If the owner accepts the request, the consumer-registration policy, which you create, is executed. This policy includes the “Register Consumer” action, which performs the actual registration process. It can optionally include other actions, such as an approval action, as needed. For additional information about creating a policy that registers consumers, see ***The Consumer Registration Policy***.

Using Policies to Manage the Deployment of Virtual Services

If you use virtual services, you must create policies that enable an administrator to deploy a virtual service to webMethods Mediator and to make a virtual service undeployable while it is being revised. For additional information about creating these kinds of policies, see ***Defining a Lifecycle Path that Enables Deployment of a Virtual Service***.

Issues to Consider When Developing Design/Change-Time Policies

The following are issues to keep in mind when creating design/change-time policies for your CentraSite registry.

- When an event occurs in the registry, CentraSite executes *all* policies whose scope encompasses the event. Use priorities to control the order in which CentraSite executes the policies. Consider assigning something other than the default priority of 11 to routine policies. Doing this will enable you to more easily interject higher priority policies for the events associated with those policies, should you ever need to do so.
- Many of the built-in actions provided with CentraSite (including the approval actions) are scoped for state-change events and can only be used with objects that are under lifecycle management. Before you create a policy, determine which policy actions you want to use and check the event types that they support. To create certain types of policies, you might need to first apply a lifecycle model to the objects that you intend to govern with those policies.
- *Exercise caution when using OnTrigger policies!* If the policy updates objects, be certain that the scope of the policy is set precisely and targets only the set of objects that you intend to change. Used inappropriately, this type of policy can change objects in ways that cannot easily be undone.
- By default, CentraSite is configured to record only policy failures in the policy log. You can optionally configure CentraSite to log both successful and failed policies in its log. However, if you do this, the log can grow quite rapidly. Consider logging successful policy executions only

when it is necessary for tracing or troubleshooting purposes. Or, if you choose to enable this option as part of your normal operations, consider purging the log on a regular basis.

- When CentraSite executes a policy, the policy's actions are executed on behalf of the user who triggered the policy (in other words, the actions are executed under that user's account). If the user does not have the permissions necessary to complete an operation initiated by a policy action, the action (and thus, the policy) will fail. For example, if a user triggers a policy that sets permissions on an object, the policy will fail unless the user has Full permission on the object. (Only users with Full permission on an object are allowed to change the object's permission settings.)

Managing Design/Change-Time Policies

The following sections describe issues relating to the creation and maintenance of design/change-time policies.

Activating a Design/Change-Time Policy

Policies are managed by a predefined lifecycle installed with CentraSite. This lifecycle, called the “Policy Lifecycle”, defines the following lifecycle states: New, Productive, Suspended and Retired.

To activate a policy, you must place the policy in the Productive state. When you switch the policy to this state, CentraSite begins enforcing it.

Modifying a Design/Change-Time Policy

You cannot modify a policy while it is in the Productive state. To make changes to a policy, you can do any of the following:

- Create a new version of the policy, make the necessary changes to the new version and switch the new version to the Productive state when you are ready to put it into effect. Switching the new version to the Productive state will immediately put the previous version in the Retired state. (The Retired state is an end state. After you place a policy in this state, you can no longer reactivate it.)
- Create a completely new policy that includes the required changes. When you are ready to put the new policy into effect, switch the old policy to the Suspended state and switch the new policy to the Productive state. When you are certain that you will no longer need to revert to the original policy, switch it to the Retired state.
- Switch the existing policy to the Suspended state, make the necessary changes to the policy and then switch it back to the Productive state. While the policy is in the Suspended state, it will not be enforced. (Because suspending the policy results in an enforcement gap, one usually does not use this approach in a production environment.)

Copying a Design/Change-Time Policy

A design/change-time policy can become quite complex, especially if it contains several policy actions. Instead of creating a new policy “from scratch”, it is sometimes easier to copy an existing policy that is similar to the one you need and edit the copy.

CentraSite includes a copy feature that lets you do this. It produces a copy that is identical to the original policy. Unlike a new version of a policy, a copy of a policy is not associated with the original policy in any way. CentraSite treats the copy just as if it were a new policy that you created from scratch.

Modifying the Predefined Lifecycle Model for Policies

The predefined lifecycle model that CentraSite uses for policies is made up of four states: New, Productive, Suspended and Retired. This lifecycle is generally adequate for most environments. However, you can make certain minor types of customizations to it if necessary.

For information about the ways in which you can customize this lifecycle model, see the topic *Customizing the Predefined Lifecycle Model for Policies* in the section *Working with Design/Change-Time Policies > Functional Scope* .

11 Working with Versions and Revisions

CentraSite's versioning capabilities enables you to maintain multiple versions of the same object.

What is Versioning?

Versioning an object creates a clone of the object. When you version an object, CentraSite copies the source object with all of its attributes and then increments the version number in the “cloned” target object. Additionally, CentraSite establishes a “supersedes” relationship between the new version of the object and the old one. Among other things, this relationship enables you to view and manage all versions of an asset from the asset's **Versions** profile in CentraSite Control.

Other than the differences described above, the new version is treated just as though it were a new object. If it is of a type that is governed by a lifecycle model, the new version of the object enters the *initial state* of that model (just like any new object of that type). Ownership of the new version is given to the user who generates the new version. Consequently, the new version of the object becomes part of that user's organization. Additionally, the instance-level permissions for the new version are reset (the permission settings from the source object *are not* copied to the new version).

Which Objects Can You Version?

You can create versions of the following types of objects:

- Design/Change-Time Policies
- Run-Time Policies
- Lifecycle Models
- Assets (except Virtual Services; see ***Revising a Virtual Service***)

Whether a particular type of asset can be versioned depends on whether the **Enable Versioning** property is enabled for the type. To version an asset, this property must be enabled in the asset's type definition.

What is a Revision?

Revisioning is an optional feature that you enable or disable for the registry. When you enable the revisioning feature, CentraSite automatically generates a minor version of an object every time an update is made to the object. (All previous revisions are retained when a new one is created.) Revisioning enables you to reference or revert back to a specific revision of an object.

Note: When you enable the revisioning feature, CentraSite maintains revisions for all objects in the registry, not just objects that are versioned.

In some API-related documentation for CentraSite, a revision is referred to as a *checkpoint.*

System-Assigned Version Identifiers vs. User-Assigned Version Identifiers

CentraSite maintains two version identifiers for a versioned object: a *system-assigned identifier* and a *user-assigned identifier*.

- The system-assigned identifier is a 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 numeric and always has the format *MajorVersion.Revision*. The *MajorVersion* number is incremented by one each time you create a new version of the object (e.g., 1.0, 2.0, 3.0). If the revisioning feature is enabled for your registry, the *Revision* number is incremented when updates are made to the object (e.g., 1.0, 1.1, 1.2). You cannot delete or modify an object's system-assigned identifier.
- The user-assigned identifier is an optional identifier that you can assign to a specific version of an object. This identifier does not need to be numeric. For example, you might use values such as “V2 Alpha” or “V2.1 Build 0921” to identify a particular version of an object.

Using Versions to Avoid Enforcement Gaps in Policies and Lifecycle Models

CentraSite does not permit you to modify a policy (design/change-time *or* run-time) or a lifecycle model while it is active. To modify an active instance of these types of objects, you must first deactivate it. However, doing this results in a period of time during which the policy or lifecycle model will not be enforced.

To avoid a gap in enforcement, you can use the versioning feature to modify these types of objects. Instead of deactivating the existing policy or lifecycle model, you generate a new version of the object and make your changes to the new version. When you are ready to put the updated policy or lifecycle model into effect, you simply switch the new version to the “Productive” state. When you do this, CentraSite automatically deactivates the old policy or lifecycle model and activates the new one.

For more information about versioning policies and lifecycle models, see ***Modifying a Design/Change-Time Policy***, ***Modifying a Run-Time Policy*** and ***Versioning or Replacing a Lifecycle Model***.

When Should You Version an Asset?

It is not necessary to version an asset to make minor changes to its metadata (e.g., the addition of a supporting document, a change in the Description attribute, the addition or removal of a particular classifier or association). Nor should you generate versions for each incremental revision of an asset that is produced during its development cycle.

You should only add a new version of an asset to the registry when you have a new implementation of the asset that is incompatible with the asset's existing consumers. For example, if a provider updates (or intends to update) a Web service in a way that “breaks” the existing applications that use the service, then you should create a new version of the asset in the registry. Such changes would include modifications to the namespace assignments, message descriptions, interface definitions and/or operation signatures in the service WSDL. It would also include changes to the implementation of the service that do not explicitly affect the WSDL, but nevertheless affect the way in which an existing consumer application interacts with the service. For example, a service that returns an expanded set of result codes or generates a different form of customer ID might break an existing consumer application even if the interface defined in the service WSDL did not change. In short, you want to create a new version of a service asset for any new implementation of the service that is incompatible with consumers of the existing service.

Note: Be aware that sometimes versioning one asset will necessitate the versioning of another. For example, if an XML schema changes, and that schema is imported by a Web service, you will need to generate a new version of the XML schema and a new version of the Web service that references it.

If a provider creates an implementation of a service that does not introduce compatibility issues with consumers of the existing service (e.g., bug fixes or performance improvements), you do not need to generate a new version of the service asset in CentraSite. Likewise, you do not necessarily need to version a service if a provider simply adds operations to the existing implementation. You can represent this type of change by just updating the existing asset in the registry.

How Do Lifecycles and Versioning Relate?

When you version an asset that is governed by a lifecycle model, the new version of the asset enters the initial state of the lifecycle. If necessary, you can modify this behavior using design/change-time policies.

For example, if the lifecycle model for an asset type includes preliminary steps such as Analysis, Planning and Architectural Design, you might want to bypass these steps when a new version of an asset is created. You can do this by executing a design/change-time policy that switches a new version of an asset to an alternate path in the lifecycle model. For details, see ***Creating a Different Lifecycle Path for a New Version of an Asset***.

Managing the Transition to a New Version

When you deploy a new version of a Web service, you must plan for transitioning existing consumers to the new version. The approach you use depends on how you intend to add the new service to your existing operational environment. You might, for example, choose to host both the old and new versions of the service for a period of time. Or you might require existing consumers to upgrade to the latest version of the service, and then, at an agreed upon hand-over date, replace the existing service with the new one. In either case, you should take steps to proactively notify existing consumers when a new version begins its development lifecycle so that they can take steps to adapt to the changes as necessary. You can use CentraSite to identify the existing consumers of the service and notify the consumers manually or you can create policies to do this automatically when a new version reaches a particular state in its lifecycle.

After a new version of an asset is placed into production, new consumers should be discouraged from using the older versions of the asset. Applying a lifecycle model that includes the "Deprecated" state is a good way to indicate to users that an asset has been replaced by a newer version. As a best practice, you should always switch an older version of an asset to the deprecated state as soon as a new version of the asset is placed into production.

Versioning in CentraSite vs. Versioning in Source Code Control Systems (SCCS)

An instance of an asset in CentraSite represents the finished deliverable that was produced (or will be produced) by a specific development project or release cycle. CentraSite is not intended to be used to chronicle every incremental build or internal version of an asset that a team of developers generates during a development cycle. However, when you register an asset in CentraSite, you might want to include in its metadata an attribute that identifies the asset's corresponding set of files in the source-control system. For example, when you place a service in production, you might want to attach a build number, branch ID or submission label to the asset in CentraSite to identify the specific set of source-code files that the asset represents. You could define a custom attribute to hold this identifier or you could use the user-assigned version identifier for this purpose.

Cleaning Up Old Versions

To keep the registry at a manageable size and reduce clutter, you should delete older versions of assets when they are no longer active. To do this, you can use the purge feature that is provided on the asset's **Version** profile. (If you do not see the **Version** profile for an asset, you will need to edit the asset's type definition to enable it.)

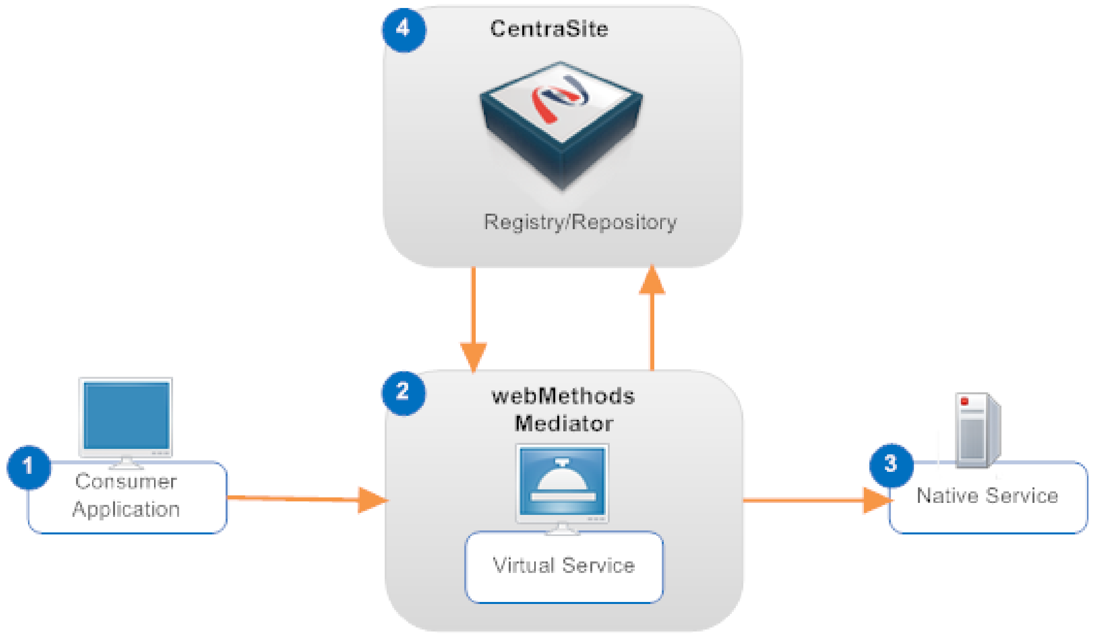
12 Planning Your Run-Time Environment

The following sections contain information that you should review if you intend to use CentraSite to govern Web services at run time.

Basic Components in the Run-Time Environment (when using webMethods Mediator as the PEP)

When you use webMethods Mediator as your policy-enforcement point (PEP), your run-time environment consists of four main components: consumer applications, webMethods Mediator, native services and CentraSite.

Basic Components in the Run-Time Environment



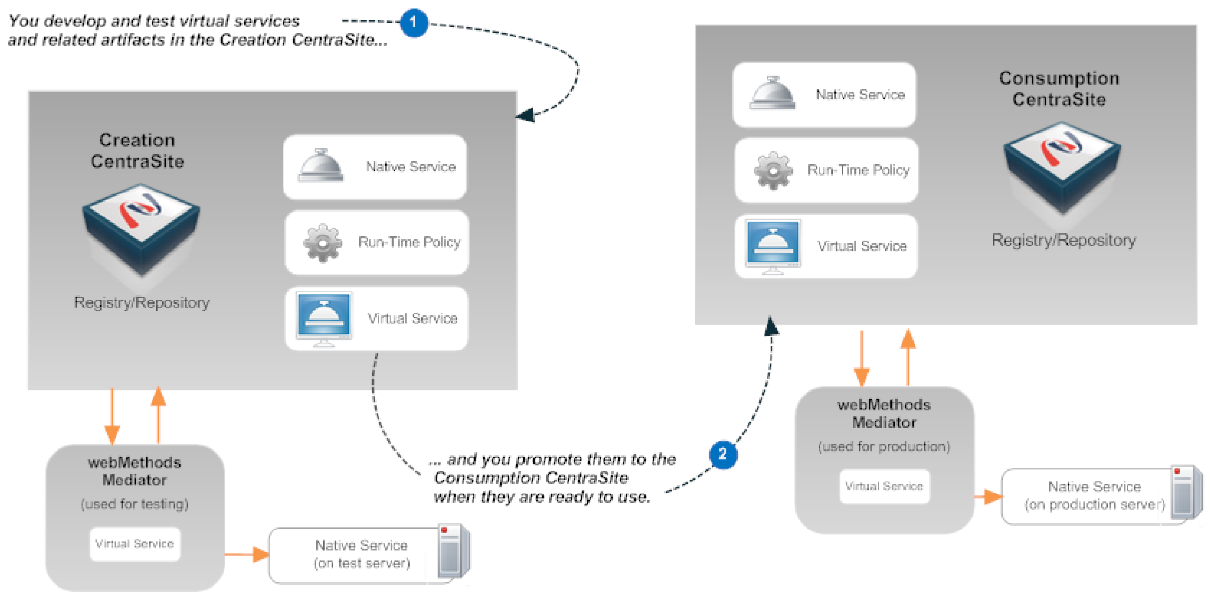
1	*Consumer applications* submit requests for operations that are provided by Web services that reside on various systems in the network. As shown in the figure above, a consumer application submits its request to a virtual service on the webMethods Mediator and not directly to the Web service itself.
2	*webMethods Mediator* hosts virtual services, which are proxy services that receive requests from consumer applications on behalf of a particular Web service. A virtual service enforces standard policies that you define for your environment (such as security enforcement and audit-trail logging) and handles mediation measures between consumer and provider such as protocol bridging, message transformation and message routing.

	Besides serving as an intermediary between consumer applications and native services, the Mediator also collects performance statistics and event information about the traffic flowing between consumer applications and the native services and reports this data to CentraSite.
3	*Native services* are Web services that process requests submitted by consumer applications. If a native service produces a response, it returns the response to the virtual service and the virtual service returns it to the consumer application.
4	*CentraSite* serves several key roles in the run-time environment. Besides serving as the system of record for virtual services, run-time policies and related artifacts in the SOA environment, CentraSite provides the tools you use to define virtual services and deploy them to webMethods Mediator. Additionally, CentraSite receives and logs the performance metrics and event data collected by webMethods Mediator and provides tools for viewing this data.

Deploying CentraSite for Run-Time Governance

When you use CentraSite for run-time governance, we suggest that you use a two-stage deployment like the one shown below. Note that with this deployment strategy, *each instance of CentraSite has its own webMethods Mediator*. This configuration creates an "air gap" between the development and production environments, which completely separates the components that support your production applications and services from those that support development and testing.

Note: For more information about the two-stage deployment strategy, see *Choosing a Deployment Strategy*.



An Overview of the Creation CentraSite Run-Time Environment

The creation environment supports the development and testing of run-time policies and virtual services. It is used by the following types of users:

- *Developers, analysts, or other authorized CentraSite users* publish the native services that will be developed and added to the SOA environment.
- *Policy administrators* develop and test the run-time policies that are to be applied to the native services when they are virtualized.
- *Asset administrators* create and test the virtual services that will be used to mediate access to the native services.

After a virtual service has been created and tested on the creation CentraSite, the virtual service, the run-time policies associated with it and the native service that it represents are promoted to the consumption CentraSite.

An Overview of the Consumption CentraSite Run-Time Environment

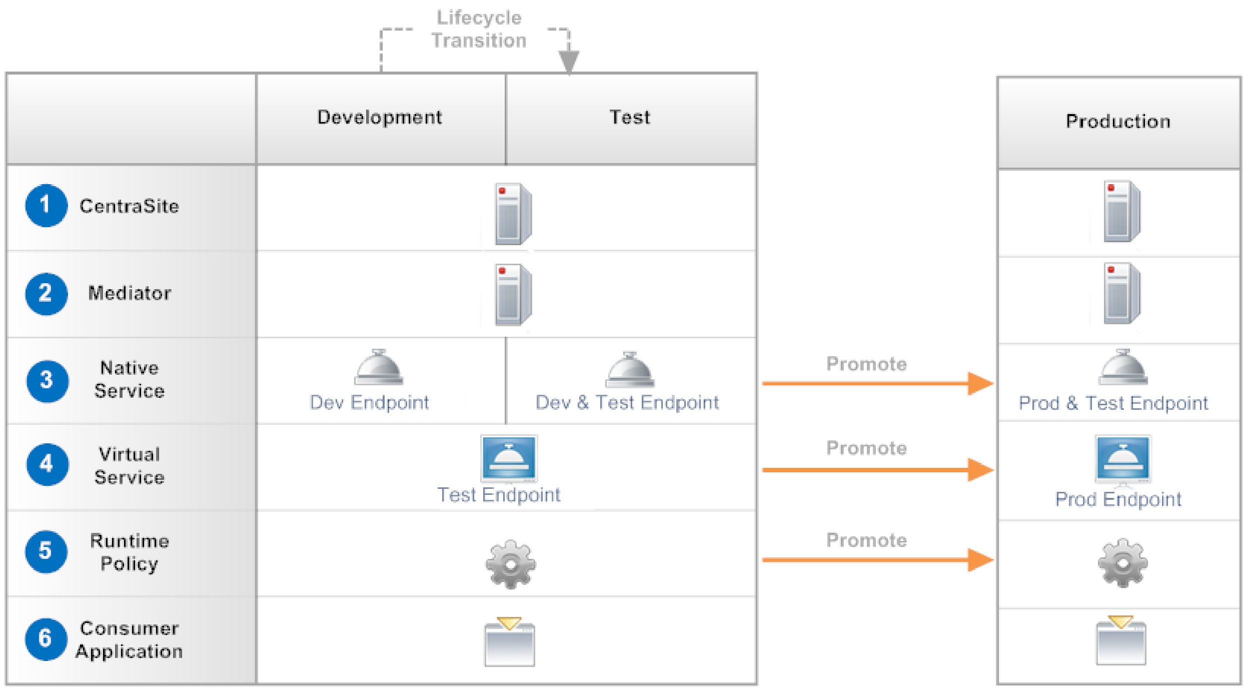
The consumption CentraSite supports the production environment. Typically, the consumption CentraSite is managed by the Operations or IT organization and only users in this organization are permitted to publish assets to it.

The consumption CentraSite is used by the following types of users:

- *Developers and analysts* access the consumption CentraSite to discover services that are available for reuse. When access to a native service is mediated by a virtual service, users who browse the catalog for re-usable services will see the virtual service, not the native service itself.
- *Designated administrators* from the IT or Operations organization import the virtual services, run-time policies and native services that have been developed and tested in the creation CentraSite. These administrators will also:
  - Adjust the permissions settings to ensure that these objects can be viewed by the appropriate groups of users (for example, they typically hide native services from developers and analysts who browse the catalog for reusable services and expose only the virtual services to those users).
  - Deploy virtual services to the Mediator.
- *Owners of the consumer applications* that invoke virtual services access the consumption CentraSite to view the performance metrics and other events relating to the operation of virtual services running in the Mediator.

General Concept of Operations between the Environments

The following diagram illustrates how the creation and consumption instances of CentraSite interrelate.



#	Description
1	*CentraSite.* The creation CentraSite supports the development and testing of services and virtual services, and the consumption CentraSite supports services and virtual services that are in production. Typically, both registries have the same basic organizational structure, although they each might each have certain "utility" organizations that are unique to their role as a creation or consumption server. (Note that the two instances of CentraSite are not required to have the same organizational structure. They can have different structures if that approach better suits your needs.)
2	*Mediator*. Each instance of CentraSite has its own Mediator (or Mediators). The Mediator in the creation environment provides a test bed that developers use during the development of virtual services and run-time policies. The Mediator in the consumption CentraSite is used exclusively for hosting virtual services that are in production.
3	*Native Services*. A native service begins its lifecycle on the creation CentraSite. When the service is ready for production, you promote it to the consumption CentraSite. On the consumption CentraSite, the native service is typically hidden from users who browse the catalog looking for services to reuse and is visible only to certain administrators (as a best practice). Note that the catalog entry for a native service includes the endpoint(s) where the service is deployed. As indicated by the figure above, these endpoints evolve as the service moves through development, test and production. The handling of endpoints is discussed in more detail in *Managing Endpoints*.

#	Description
4	*Virtual Services*. Like a native service, a virtual service begins its lifecycle on the creation CentraSite. You cannot create a virtual service until the native service it represents is registered in the creation CentraSite and it has been deployed on an endpoint in the network. Typically, one creates a virtual service during the late stages of the development phase or when the native service enters the test phase (in other words, after the service's interface is completely implemented and an instance of the service is deployed and running at a known point in the development environment). After the virtual service has been tested and it is considered ready for production use, it is promoted to the consumption CentraSite and deployed to a production Mediator.
5	*Run-time Policy*. A run-time policy defines a sequence of standard policy-enforcement actions that are to be executed when a virtual service is invoked. Administrators define and test run-time policies on the creation CentraSite. When the policies are considered ready for production use, they are promoted to the consumption CentraSite. **Note:** Before you deploy a virtual service to the Mediator in the consumption environment, it is important to ensure that all the run-time policies that apply to the virtual service have been promoted to the consumption CentraSite.
6	*Consumer Application*. A consumer application identifies an application that invokes virtual services. Consumer applications are defined directly on the consumption CentraSite. They are not promoted from the creation CentraSite. (Administrators on the creation CentraSite who define run-time policies will typically define dummy consumers for testing purposes.) **Note:** Technically speaking, a consumer application is represented by an Application asset in the registry. For more information about defining consumer applications, see ***Identifying the Consumers of Virtual Services***.

13 Implementing the Mediation Environment

To use an instance of CentraSite with webMethods Mediator, you must define a *target* that identifies the specific webMethods Mediator that you want to use. A target is a registry object that represents a particular instance of a policy enforcement point (in this case, an instance of webMethods Mediator). The target object specifies the address of the Mediator's *deployment endpoint*, which is the endpoint that CentraSite uses to interact with Mediator to deploy virtual service.

If you use multiple Mediators with an instance of CentraSite, you must create a target for each Mediator. To make the Mediators easier to distinguish when they are viewed in CentraSite, consider adopting a naming convention for targets that clearly identifies to which environment the target belongs (e.g., development, test, production).

Targets are defined by an administrator with "Manage Runtime Targets" permission. Targets are system-wide objects that are shared by all the organizations within an instance of CentraSite.

To communicate with CentraSite, a Mediator must have a user account on CentraSite. You have to establish this user account before you can configure Mediator's connection to CentraSite. Because Mediator reads and writes to certain objects in the registry (such as virtual services), its user account requires a specific set of permissions. These permissions are identified in the Mediator user documentation.

Note: If you expect a high volume of traffic through a particular Mediator, consider clustering that Mediator. When you cluster Mediators, the cluster is represented by a single target within CentraSite.

Managing the Collection of Metrics

webMethods Mediator collects performance data (e.g., average response time, total request count, fault count) for the virtual services that it hosts. It publishes this data to CentraSite at regular intervals. When you install and configure the Mediator, you must specify whether you want it to collect performance data and, if so, how often you want it to publish the data to CentraSite.

We recommend that you always enable the collection of performance data on your Mediator. A publication interval of 15 minutes is appropriate for most environments. However, if the Mediator will handle a very high volume of traffic, consider increasing this interval to 30 or 60 minutes. CentraSite stores the performance data that it receives from the Mediator in the performance log. You can look at the performance information for a particular virtual service by viewing the virtual service's **Performance** profile.

The performance data that CentraSite collects from Mediator can cause the log to grow quite rapidly. When the log grows very large, queries to the log can significantly affect CentraSite's performance. To prevent this from happening, we suggest that you routinely purge old entries from the log.

CentraSite provides a log-purging utility that you can use to automatically purge the log on a scheduled basis. We suggest that you use this utility to keep no more than one month of performance data in the log (adjust this recommendation as necessary to accommodate your particular needs). When you configure the log-purging utility, you can specify whether you want to delete the purged log entries or export them to an archive file (in case you want to retain them for future reference).

Note: The performance metrics that Mediator collects enable service consumers (and potential service consumers) to determine whether a virtual service is performing at a required level. Mediator does not collect data at the granularity that a network administrator would need in order to analyze performance problems (e.g., to determine why the response time for a particular virtual service drops at a particular time of day). If you need data to support this level of analysis, consider adding an Insight agent to your Mediator.

Managing the Collection of Events

In addition to performance metrics, webMethods Mediator can also log *event data*. Event data supplies information about activities or conditions that occur on Mediator.

Mediator logs two basic kinds of events: 1) data relating to the operation of Mediator itself and 2) data relating to the execution of virtual services.

The event data that Mediator collects about itself are referred to as *lifecycle events*. These events represent activities or conditions that occur during the general operation of Mediator. Lifecycle events are reported for the completion of significant processes (e.g., Mediator start-up) and the detection of operational exceptions and policy violations. Mediator logs lifecycle events if you configure it to do so.

Mediator collects the following types of event data relating to the execution of virtual services. Be aware that Mediator does not collect this type of information automatically. If you want to capture these types of events, you must deploy run-time policies to do so.

- *Transaction Events* report information about the requests that the Mediator processes. This type of event is produced by the execution of a logging action in a run-time policy. For example, you might configure a run-time policy to log all of the request and/or response messages submitted to a particular virtual service.
- *Monitoring Events* report transgressions relating to performance metrics. This type of event is produced by the execution of a monitoring action in a run-time policy. For example, you might configure a run-time policy to report occasions when the response time for a virtual service exceeds a specified threshold.

Mediator publishes event data as Simple Network Management Protocol (SNMP) traps. When you install Mediator, you can configure it to publish this data to CentraSite, to another third-party SNMP server or to both. If you choose to log event data to CentraSite, you can view the events using the CentraSite Control user interface.

Note: If you will be logging event data to CentraSite, you must first configure CentraSite's event listener as described in the section *Run-Time Governance Reference > Run-Time Events and Key Performance Indicator (KPI) Metrics* .

Like the performance log, the event log will grow larger over time. If it becomes very large, queries to the log will cause performance issues with CentraSite. To manage the size of the event log, we suggest that you occasionally purge old entries from it. As a general guideline, consider maintaining only three months of event data in the log. (Adjust this recommendation as necessary to accommodate your particular needs. If you routinely log request and response messages, for example, you might need to purge more often.)

You can configure CentraSite's log-purging utility to purge the event log on a scheduled basis. When you configure the purging facility, you can specify whether you want to delete the purged entries or export them to an archive file (in case you want to retain them for future reference).

Using CentraSite with Other Policy Enforcement Points

Instead of (or in addition to) using webMethods Mediator for mediation and/or policy enforcement, you can use other third-party products with CentraSite. Support for third-party policy-enforcement and run-time governance tools is available through integrations that are provided by members of the CentraSite Community. These tools are made available through the CentraSite Community Web site at **http://www.centrasite.org**.

Using CentraSite with Insight

Insight is an additional monitoring tool from Software AG that you can use with CentraSite. Insight provides deep visibility and monitoring of services in a heterogenous SOA environment. CentraSite provides support for Insight as a target type out-of-the-box. For additional information about Insight's uses and capabilities, see the Insight user documentation.

14 Managing Run-Time Policies

A run-time policy defines a sequence of actions that a policy-enforcement point (PEP), such as webMethods Mediator, will carry out when the PEP receives a request from a consumer application. The actions that a run-time policy can execute depend on the type of PEP you are using. If you are using webMethods Mediator, for example, you might create run-time policies to perform the following kinds of tasks:

- Verify that the requests submitted to a virtual service come from consumer applications that are authorized to use the virtual service.
- Validate request and response messages against the schema specified in the service's WSDL.
- Log the request and response messages to the CentraSite event log.
- Alert an administrator if the average response time for a service drops below a specified threshold.

Run-time policies enable service architects to separate the logic that relates to general functions, such as security, message validation, logging and monitoring, from the business logic of the service. By using policies to carry out these general activities, an organization can easily modify and re-deploy the policies as necessary without disrupting the native services that perform the business logic.

Like a design/change-time policy, a run-time policy consists of two major elements: an *action list* and a defined *scope*. The policy's action list specifies the sequence of actions that are to be executed by the PEP. The policy's scope determines to which virtual services the policy applies.

Note: The remaining sections assume that you are creating run-time policies that will be deployed with virtual services on webMethods Mediator. If you are using another PEP, the principles described in these sections will be similar, however, it will support its own unique set of policy actions. *The built-in policy actions that CentraSite provides out-of-the-box are to be used only with webMethods Mediator. You cannot use these actions with other PEPs.*

Run-Time Policy Scope

The scope of a run-time policy specifies for which virtual services the policy will be enforced. The scope of a run-time policy is determined by the following policy parameters:

- *The Target Type* parameter specifies the type of PEP on which the policy will be enforced. When you create run-time policies for webMethods Mediator, you set the Target Type to “webMethods Integration Server”.
- *The Organization* parameter specifies to which organization's virtual services the policy applies. Like design-time policies, a run-time policy can be system-wide or organization-specific. System-wide policies apply to virtual services in any organization. Organization-specific policies apply only to virtual services that belong to a specified organization.

Note: Once you create a policy, its organizational scope is fixed and cannot be changed. That is, if you create a policy whose scope is specific to organization ABC, you cannot change its scope to make it system-wide or switch it to another organization. You must create a new policy and set its organizational scope as needed.

- *The Asset Type* parameter specifies to which types of assets the policy applies. When you create run-time policies for webMethods Mediator, you set the **Asset Type** parameter to "Virtual Services". When you are creating a run-time policy for another type of PEP, you will set this parameter to "Web Services" (or possibly to some other asset type).

Refining the Run-Time Policy's Scope with Additional Selection Criteria

Although it is possible to create a run-time policy that applies to all virtual services, this is not the common case. More frequently, you will create run-time policies that apply to a particular set of virtual services (or possibly even one specific virtual service). For example, you might define a run-time policy that monitors the response time for all virtual services that are considered to be "critical".

To target a policy for a particular set of virtual services, you refine the policy's scope by specifying additional selection criteria based on the virtual service's Name, Description or Classification properties. For example, if you wanted to apply a particular run-time policy to "critical" services as described above, you would classify virtual services according to their criticality and create a policy that targets virtual services that are classified as "critical".

In cases where you need to apply a run-time policy to one specific virtual service, you can use the selection criteria to identify the virtual service by name.

Note: While you are creating a run-time policy, you can refer to the policy's **Service** profile to see exactly which set of virtual services are currently within the policy's scope.

To use run-time policies effectively, you need to think about what selection criteria your policies will use to identify the set of virtual services with which they are to be used. You must also ensure that the virtual services in your registry adhere to the Name, Description and/or Classification conventions needed to support your selection scheme. For example, if you intend to enforce different logging policies for different classes of virtual services, you must define the taxonomy by which the virtual services will be classified (for logging purposes) and ensure that virtual services are classified according to this taxonomy before they are deployed.

Run-Time Policy Actions

CentraSite is installed with a set of actions that you use to define run-time policies for virtual services deployed on webMethods Mediator. These actions fall into the following four categories:

Security actions, which you use to identify and authenticate the consuming application that submitted a request, to enforce the use of the SSL protocol (if required), to perform encryption and decryption of specified parts of the request and response messages, and to validate signatures of messages that are digitally signed.

Logging actions, which you use to log the request and/or response messages associated with a virtual service. Logging actions can send the logged messages to Mediator's local log, CentraSite's policy or audit log, to an SNMP server and/or email the messages to specified addresses.

Performance monitoring actions, which you use to monitor specified metrics (e.g., service availability, average response time, fault count, request count) and log instances when these metrics violate specified thresholds. Violations can be reported in Mediator's local log, in CentraSite's policy or audit log, in an SNMP server and/or sent as an email message to specified addresses.

Data validation actions, which you use to validate the request and/or response message against the WSDL associated with the virtual service.

Note: CentraSite does not require you to deploy run-time policies with a virtual service. It is possible to deploy virtual services to webMethods Mediator without accompanying policies. In practice, however, this is rare. One nearly always deploys virtual services with, at the very least, some type of security-related policy to identify and authenticate the consuming application.

Run-Time Policy Deployment

When you deploy a virtual service to webMethods Mediator, CentraSite combines the actions from all of the run-time policies that apply to the virtual service and generates what is called the *effective policy* for the virtual service. For example, let's say your virtual service is within the scope of two run-time policies: one policy that performs a logging action and another policy that performs a security action. When you deploy the virtual service, CentraSite automatically combines the two policies into one effective policy. The effective policy, which contains both the logging action and the security action, is the policy that CentraSite actually deploys to the Mediator with the virtual service.

When CentraSite generates the effective policy, it validates the resulting action list to ensure that it contains no conflicting or incompatible actions. If the list contains conflicts or inconsistencies, CentraSite resolves them according to the rules described in the section *Run-Time Governance Ref-*

erence > Built-In Run-Time Actions Reference for Virtual Services . For example, an action list can include only one Identify Consumer action. If the resulting action list contains multiple Identify Consumer actions, CentraSite resolves the conflict by including only one of the actions (selected according to a set of internal rules) in the effective policy and omitting the others.

The effective policy that CentraSite produces for a virtual service is contained in an object called a *virtual service definition* (VSD). The VSD is given to Mediator when you deploy the virtual service. After you deploy a virtual service, you can view its VSD (and, thus, examine the effective policy that CentraSite generated for it) from CentraSite Control or from the Mediator user interface.

Creating and Testing Policies

In a two-stage deployment of CentraSite, you will create and test run-time policies on the creation CentraSite and then promote them to the consumption CentraSite when they are considered ready for use.

Distributing the Development of Policies

Because run-time policies involve several different aspects of operational behavior, there are often several different types of architects or experts involved in their creation. For example,

- *Security-related policies* might be defined and managed by a security architect, who is responsible for ensuring that the policies adhere to corporate security standards.
- *Performance-monitoring policies* might be defined and managed by an administrator or analyst from the Operations organization.
- *Application-related policies* (i.e., policies that involve logging or validating the data associated with the service itself) might be defined and managed by an SOA Application Architect.

Because CentraSite combines applicable run-time policies at deployment time, it is possible to distribute the policy-development responsibilities to the appropriate experts in your organization as described above. Each expert defines and manages the policies for his or her area of expertise. Then, at deployment time, CentraSite combines the applicable policies produced by each of these experts into one effective policy for the virtual service.

Activating a Run-Time Policy

A run-time policy is not eligible for deployment unless it is “active”. In other words, when CentraSite deploys a virtual service, it uses only “active” run-time policies to produce the effective policy for the virtual service. Whether a run-time policy is active or inactive is determined by its lifecycle state.

Run-time policies are governed by the same lifecycle model as design/change-time policies. The predefined lifecycle model for policies that is installed with CentraSite is made up of four states: New, Productive, Suspended and Retired. Under this lifecycle model, all policies enter the New state when they are initially created.

While a run-time policy is in the New state, it is inactive. To activate a run-time policy, you must place the policy in the Productive state. Switching the policy to the Productive state triggers an internal system policy that activates the policy and makes it eligible for deployment.

If you activate a run-time policy that applies to a virtual service that is already deployed on webMethods Mediator, be aware that *CentraSite will automatically re-deploy that virtual service with the updated run-time policy*. That is, CentraSite will automatically regenerate the effective policy for the virtual service (to include the newly activated run-time policy) and then redeploy the virtual service on webMethods Mediator.

To deactivate a run-time policy, you switch the policy to the Suspended state or the Retired state. When you deactivate a run-time policy, *CentraSite will automatically remove the policy from any virtual service with which it is currently deployed*. That is, CentraSite will automatically regenerate the effective policy for each of those virtual services (to remove the deactivated policy) and then redeploy the virtual services on webMethods Mediator.

Note: We recommend that you use the lifecycle model that CentraSite provides for policies “as-is”, even for a multi-stage deployment. For more information, see ***Lifecycle Model for Policies in a Multi-Stage Deployment of CentraSite.***

Modifying a Run-Time Policy

You cannot make changes to a run-time policy while it is active. To make changes to a policy after it has been switched to the active state you must do one of the following:

1. Switch the policy to the Suspended state (to deactivate it), update the policy and then switch it back to the Productive state (to reactivate it).

   OR

2. Create a new version of the policy, make your changes to the new version of the policy and then switch the new version to the Productive state. Switching the new version of the policy to the Productive state will automatically Retire (and deactivate) the old version.

If you need to update a run-time policy that is already deployed with virtual services that are in production, always use the second method described above (i.e., create a new version of the policy). If you use the first method, which requires you to suspend the existing policy, your production services will be running without the policy while you are making your revisions to it.

Additionally, if you need to make a change to a run-time policy on the creation CentraSite, and that change needs to be promoted to the consumption CentraSite, always create a new version of the policy and then promote the new version to the consumption registry. This will ensure that the import process creates a new instance of the policy on the consumption registry (i.e., that it does not overwrite the existing run-time policy) and that the updated policy is added to the registry in the New (inactive) state. When you are ready to activate the imported policy, you switch it to the Productive state.

Lifecycle Model for Policies in a Multi-Stage Deployment of CentraSite

CentraSite provides a lifecycle model for policies that applies to both design/change-time policies and run-time policies. The lifecycle model has associated policies that CentraSite uses to validate, activate and deactivate policies. Because of the complex nature of this lifecycle model, we recommend that you use the lifecycle model as installed. Do not attempt to customize this lifecycle model. (You can associate additional policies with its state changes, however.)

If you are operating in a multi-stage deployment, this means that you will use the *same lifecycle model* on all stages in your environment. You will not define a multi-stage lifecycle model for policies as you would for an asset.

When you develop a run-time policy on the creation CentraSite, that policy will enter the New state. It will transition between the Productive and Suspended states as it undergoes development and testing. When the run-time policy is considered to be ready for production, you will promote it to the consumption CentraSite. Here it will also enter the lifecycle in the New state. When you are ready to activate the policy in the consumption CentraSite, you switch it to the Productive state.

Tip: To indicate that a policy has been promoted to the production environment, consider adding a comment such as “PROMOTED” or “[Moved to Production]” to the policy's **Description** property on the creation CentraSite.

15 Managing Virtualized Services

A virtualized service is a service that runs on webMethods Mediator and acts as the consumer-facing proxy for a service that runs elsewhere on the network. You can create a virtualized service for a SOAP-based Web service, a REST service or an XML service. A virtualized service provides a layer of abstraction between the service consumer and the service provider, and promotes loose coupling by providing location, protocol and format independence between the consuming application and the provider service.

For example, virtualized services enable you to:

- Move native services to other physical addresses or switch providers without affecting existing consumer applications.
- Bridge differences (e.g., transport differences, message structure differences) between the capabilities of a consuming application and the requirements of a native service.
- Block portions of a service interface from certain consuming applications (i.e., expose selected portions of the native service to certain consumers).
- Provide access to different versions of a service through a single endpoint.

You use CentraSite to define virtualized services and to deploy them on specified Mediators. After you deploy a virtualized service, you use CentraSite as a “dashboard” from which to view performance metrics and other run-time data relating to the usage of a virtualized service.

Which Services Should You Virtualize?

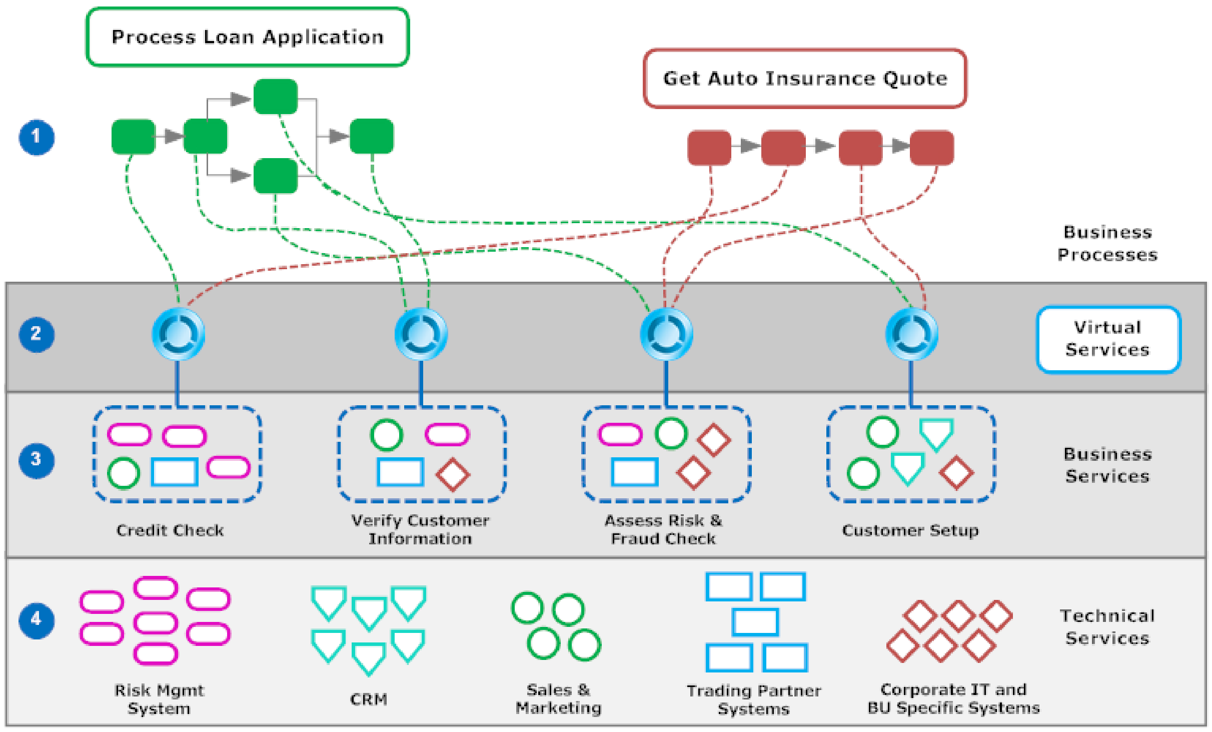
Although it is possible to virtualize any native service that is registered in CentraSite, you will generally virtualize only certain types of services. The use of virtual services creates an additional hop in the execution path and also consumes resources from an execution perspective. It is generally not practical (or beneficial) to virtualize every native service in your environment. With respect to virtualization, you want to strike a balance between the need to provide an SOA infrastructure that is flexible and extensible with the need to maintain a manageable infrastructure that is not overly complex.

Broadly speaking, there are three types of services you should consider virtualizing:

- Business services (which you virtualize at the point of consumption)
- Shared services
- Services that are provided and consumed in different domains of control (e.g., cloud computing)

Virtualizing Business Services at the Point of Consumption

As shown in the following diagram, one approach to virtualization is to think of your services in terms of *business services* and *technical services* and to virtualize those services that are business services.



#	Description
1	*Business processes* are end-user applications that perform high-level tasks within your enterprise (e.g., fulfilling an order or generating a quote). Business processes provide business functionality by orchestrating operations provided by different business services (depicted in layer 3).
2	*Virtual services* run in the layer between the business processes and business services. Each virtual service functions as a proxy for a particular business service.
3	*Business services* are coarse-grain services that perform business-related tasks, such as performing a credit check, setting up a new customer account or checking the status of an order. Business services generally perform their work by invoking the operations of many different technical services (depicted in layer 4). Because business services represent the point of consumption by end-user applications and processes, they are good candidates for virtualization. Additionally, there are generally far fewer business services than technical services (typically, 10% to 15% of services in an SOA environment are business services).
4	*Technical services* are fine-grained services that perform low-level tasks and/or utility functions such as updating the employee database, retrieving a customer record or executing a query against the order

#	Description
	database. Often, a technical service provides access to the functionality of a specific back-end system such as a CRM system, an order-entry system or a financial system. **Note:** There are cases when you might want to virtualize a technical service (see **Virtualizing Shared Services**, below). However, these situations are rare. Generally speaking, you should avoid virtualizing technical services unless they are used by consumers in multiple functional domains.

Virtualizing Shared Services

A shared service is a service that is used by multiple functional domains within an enterprise. For example, consumers in the CRM area, the Sales and Marketing area and the Risk Management area might each need access to customer data. Instead of giving these systems direct access to the data service for the customer database, you virtualize the service and give these consumers access to the virtual service (or virtual services). Virtualizing the service gives you greater control over the interface that is exposed to these consumers, enables you to accommodate differences among the consumers by applying different run-time policies and/or processing steps to them and also gives you the flexibility to make modifications to the native service without impacting existing consumers.

Virtualizing Services that are in Different Domains of Control (e.g., Cloud Computing)

Any service that is provided by an entity outside of the enterprise or is consumed by an entity outside of the organization should be virtualized. For example, if you have an outside service that provides sales forecasts for your industry, virtualizing this service would enable you to:

- Monitor the performance and availability of the service, including compliance with service-level agreements (SLAs).
- Shield consumers from changes in service providers.
- Track dependencies between the external service and the applications within the enterprise that consume the service.
- Resolve protocol and format inconsistencies between the outside service and the consuming applications within your enterprise.

Similarly, you should virtualize any service that your organization offers to applications that execute outside the enterprise (an inventory control service that you extend to suppliers and/or distributors, for example).

The Basic Elements of a Virtual Service

A virtual service is a Web service that runs on webMethods Mediator. You use CentraSite to create, edit, deploy and manage virtual services. Virtual services have the following major elements:

- *Basic service metadata and WSDL*. When you create a virtual service, the metadata from the native service is copied to the virtual service. The WSDL from a native SOAP-based Web service is also copied to the virtual service. After the virtual service is generated, you can edit its metadata and/or the WSDL as necessary.
- *A set of processing steps*. Every virtual service includes a set of processing steps that you configure before deploying the virtual service. The processing steps specify how the virtual service will handle the requests it receives from consuming applications. Processing steps are discussed in more detail later in this section.
- *One or more targets*. The **Deployment** profile for a virtual service specifies the targets (webMethods Mediators) on which the virtual service is deployed.
- *Run-time policies associated with the virtual service*. The **Policies** profile for a virtual service identifies the run-time policies that apply to the virtual service. These run-time policies are the ones that CentraSite will include when it deploys the virtual service to the Mediators specified on the **Deployment** profile.
- *Performance and Event profiles*. The **Performance** and **Events** profiles enable you to examine the run-time data associated with a virtual service. You use these profiles to view performance metrics for a specified time period and to view events that have been logged for the virtual service (e.g., SLA violations, service failures, logged request/response messages and so forth).

Virtual Service Processing Steps

The processing steps associated with a virtual service determine how the virtual service handles the requests it receives from consumer applications. All virtual services have four processing steps:

- The Entry Protocol step
- The Request Processing step
- The Routing step
- The Response Processing step

You configure these steps to specify how Mediator is to act upon the requests it receives for this virtual service.

Entry Protocol Step

The Entry Protocol step specifies the protocol (JMS, HTTP or HTTPS) in which the virtual service accepts requests. This step allows you to bridge protocols between the consuming application and the native service. For example, let's say that you have a native service that is exposed over JMS and a consuming application that submits SOAP requests over HTTP. In this situation, you can configure the virtual service's Entry Protocol step to accept HTTP requests and configure its Routing Step (described below) to route the request to the native service using JMS.

Besides using the Entry Protocol step to resolve protocol differences between the consumer and the native service, you might use this step to intentionally expose a virtual service over a particular protocol. For example, if you have a native service that is exposed over HTTP, you might expose the virtual service over JMS simply to gain the asynchronous-messaging and guaranteed-delivery benefits that one gains by using JMS as the message transport.

Request Processing Step

The Request Processing step specifies how the request message is to be transformed or pre-processed before it is submitted to the native service. You can configure this step to perform message transformations using a specified XSLT file or by passing the message to a webMethods IS service (i.e., a webMethods Integration Server service running on the same Integration Server as webMethods Mediator).

You can use this processing step to accommodate differences between the message content that a consuming application is capable of submitting and the message content that a native service expects. For example, if the consuming application submits an order record using a slightly different structure than the structure expected by the native service, you can use the Request Processing step to transform the record submitted by the consuming application to the structure required by the native service.

Routing Step

The Routing step specifies the endpoint to which requests are to be routed and the protocol (HTTP or JMS) by which they are to be submitted to the native service.

If the native service is exposed over JMS, you use the routing step to specify the queue to which the Mediator is to submit the request and the destination to which the native service is to return the response.

If the native service is exposed over HTTP or HTTPS, you can configure this step to route all requests to a specified endpoint (*straight through* routing), route requests to different endpoints based on the content of the request (*content-based* routing), route requests to different endpoints based on factors such as the time of day or the requestor's IP address (*context-based* routing) or distribute requests across multiple endpoints (*load-balancing* routing).

Note: When you configure the Routing step, you can either manually type the endpoint of the native service or you can select the endpoint from a list of known endpoints in the registry. As a best practice, you should always select the endpoint rather than typing it manually. The act of selecting an endpoint establishes a relationship between the virtual service and the native service that is hosted at the selected endpoint. This relationship is rendered when you examine the virtual service or the native service using the Impact Analysis feature.

Using the Routing Step to Direct Requests across Multiple Endpoints

If you have a native service that is hosted at two or more endpoints, you can use the load balancing option in the Routing Step to distribute requests among the endpoints or you can use the content-based or context-based options to route different types of messages to different endpoints.

Using the content-based routing option, you can route messages to different endpoints based on specific values that appear in the request message. You might use this capability, for example, to determine which operation the consuming application has requested and route requests for complex operations to an endpoint on a fast machine.

Using the context-based routing option, you can route messages based on criteria such as the time of day and/or the identity of the consuming application. For example, you might use this capability to route requests from certain high-priority consumers to endpoints on a fast machine.

Note: With either option, you must provide a default endpoint to which the virtual service can route requests that do not satisfy any of the specified criteria.

Response Processing Step

The Response Processing step is similar to the Request Processing step. This step specifies how the response message from the native service is to be transformed or processed before it is returned to the consuming application. Like the Request Processing step, you can configure the Response Processing step to perform message transformations using a specified XSLT file or by passing the message to a webMethods IS service. You can also use this step to return a customized error message to the consuming application when a SOAP fault occurs. (CentraSite provides a set of context variables that you can use to incorporate specific details about the transaction into the error message. You might use these variables to include information such as the time and date of the error, the consumer identifier, and/or the requestor's user ID.)

Configuring CentraSite for Virtual Services

Before you can create and deploy Virtual Services on your instance of CentraSite, there are two important configuration steps that you must perform.

- *You must define a target object* for each instance of webMethods Mediator that CentraSite will use.
- *You must define a lifecycle model* for services and virtual services.

Defining Targets

Before you can deploy virtual services, you must define targets to represent the Mediators that are attached to your instance of CentraSite. For example, if you will be deploying virtual services to two different Mediators, you must create two target objects, one for each Mediator instance.

Note: CentraSite will not enable the **Deploy** button on the **Deployment** profile for a virtual service until at least one target has been defined on your instance of CentraSite.

Defining a Lifecycle Model for Services and Virtual Services

To deploy virtual services, you must define a lifecycle model for services and virtual services. You must also create policies that enable and disable the Deployment profile depending on the state of the virtual service within this lifecycle. *You cannot deploy virtual services until this lifecycle model and the necessary deployment-related policies exist and have been activated.*

Note: The starter kit includes a lifecycle model for services and virtual services that you can use as a guide. For more information about the starter kit, see ***Obtaining and Installing the Starter Kit***.

Understanding the Lifecycle for Services and Virtual Services

A virtual service is a specialized form of a Service asset type. Because virtual services are actually service objects, a lifecycle model that applies to services applies to virtual services as well. Yet a virtual service and a native service have distinctly different lifecycles. To accommodate this difference, you must create a lifecycle model that defines two separate lifecycle paths.

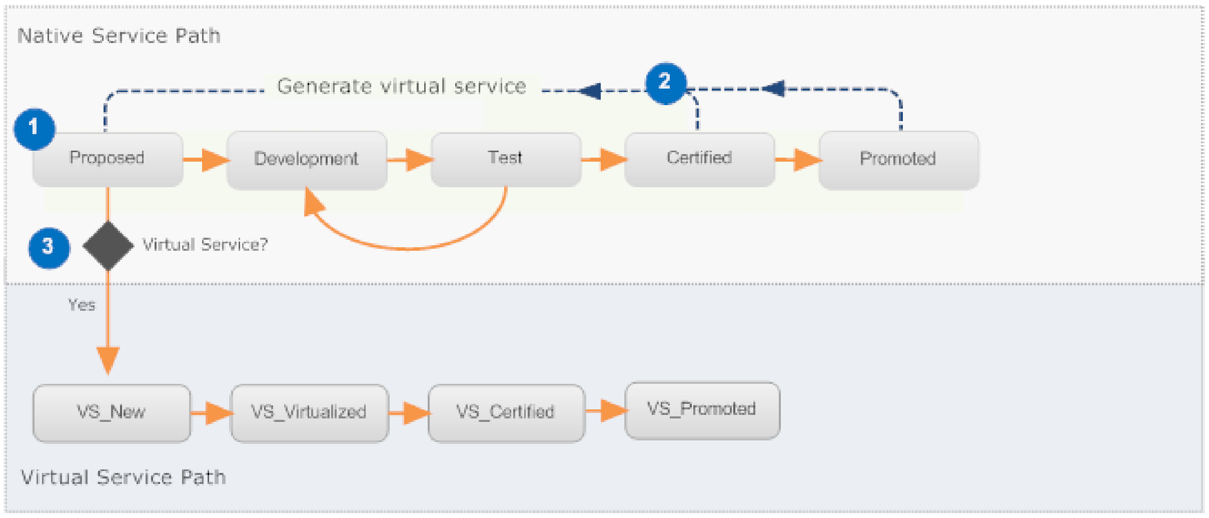
The following diagram shows a simple lifecycle model that supports both types of services. Note that this lifecycle has a path for native services and a path for virtual services. Policies are used to switch native services and virtual services to the appropriate path.

Note: To distinguish virtual services from native services (i.e., regular SOAP-based Web services, REST services or XML services), CentraSite adds the **CentraSite VirtualTypes:**

Virtual services classifier to a virtual service. This classifier enables you to create design/change-time policies that target virtual services specifically.

In the following example, the **Proposed** state is the lifecycle model's initial state. When a native service is created, it enters the Proposed state and from there, it follows the lifecycle path for native services. After a native service is tested and it is ready to be promoted for production, a virtual service is generated for it. The virtual service initially enters the Proposed state when it is created. However, a design/change-time policy immediately switches the virtual service to the lifecycle path for virtual services.

Simple Lifecycle of a Virtual Service on the Creation CentraSite



#	Description
1	The **Proposed** state is the initial state for this lifecycle model. When a native service is created, it enters the Proposed state and follows the lifecycle path for native services.
2	A virtual service is generated from the native service when the service is ready to go to production. Generally, this step is performed after the native service has been tested and is considered ready for production or after it has been promoted to the production environment.
3	The virtual service enters the lifecycle in the Proposed state, but a policy immediately switches it to the **VS_New** state, which is the beginning of the lifecycle path for virtual services. This lifecycle path includes states that enable or disable the deployment of the virtual service. **Note:** To prevent users from manually switching a native service to the lifecycle path for a virtual service, you can apply a policy to the **VS_New** state to verify that only service assets classified as **CentraSite VirtualTypes: Virtual services** enter this path.

Creating the Lifecycle Model for Services and Virtual Services

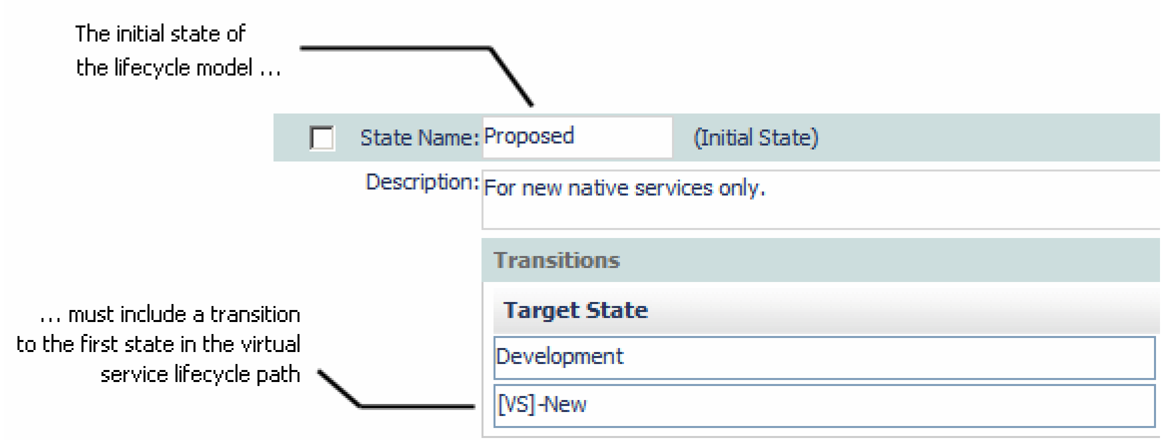
To create a lifecycle model that supports both native services and virtual services, you must perform the general steps described below.

Note: To see how the following steps have been implemented in an actual lifecycle model, install the starter kit and refer to the lifecycle model for services and virtual services. For more information about the starter kit, see *Obtaining and Installing the Starter Kit*.

1. Create a lifecycle model for the "Service" asset type and in this model define the sequence of states and transitions that make up the lifecycle path for a native service.
2. In the same lifecycle model (and following the sequence of states that you defined in the previous step), define the sequence of states and transitions that make up the lifecycle for a virtual service.

   Note: The lifecycle path for a virtual service must include at least one state that represents the point where the virtual service has been completely configured and is ready to be deployed. Before creating the lifecycle path for a virtual service, review the information in **Creating a Policy that Enables the Deployment Profile** to ensure that your lifecycle path includes the appropriate deployment-related states and policies.

3. Define a transition from the initial state of the lifecycle model to the first state in the lifecycle path for virtual services. This will be the only transition that should connect the two lifecycle paths. In the example depicted in *Understanding the Lifecycle of Services and Virtual Services*, this is accomplished by allowing a transition from the **Proposed** state, which is the initial state for the entire model, to the **VS_New** state, which is the first state in the lifecycle path for a virtual service.


The initial state of the lifecycle model ...
... must include a transition to the first state in the virtual service lifecycle path
Transitions

State Name: Proposed (Initial State)

Description: For new native services only.

Target State
Development
[VS]-New

4. Apply a policy to the lifecycle model's initial state (PostStateChange) that switches the state of a virtual service to the first state in the lifecycle path for virtual services. Use the Classification filter to scope the policy so that it executes only for virtual services.

In the example depicted in *Simple Lifecycle of a Virtual Service on the Creation CentraSite*, this policy executes on the PostStateChange for the **Proposed** state and switches virtual services to the **VS_New** state.

5. Optionally, create a policy that executes on the **VS_New** state (PreStateChange) and verifies that the service includes the CentraSite **VirtualTypes: Virtual services** classifier. Doing this will prevent someone from inadvertently switching a native service to the virtual service lifecycle path.

Defining a Lifecycle Path that Enables Deployment of a Virtual Service

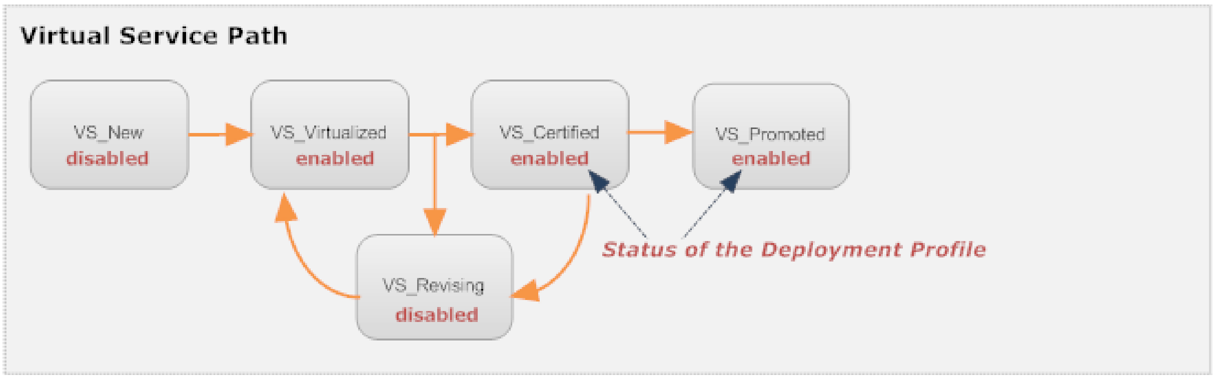
The virtual service's **Deployment** profile contains the controls that you use to deploy, undeploy and redeploy the virtual service. When the **Deployment** profile is disabled, you cannot perform these operations on the virtual service.

By default, the **Deployment** profile is disabled when you create a virtual service. This is to prevent anyone from deploying the virtual service until after its processing steps have been properly configured. To enable the **Deployment** profile, you must switch the virtual service to a state that triggers the execution of a policy that enables the **Deployment** profile.

When you define the lifecycle path for a virtual service, you must determine during which states the **Deployment** profile will be enabled and during which states it will be disabled. Then, you must create policies to enable or disable the **Deployment** profile as appropriate when the virtual service enters these states.

For example, if you wanted the **Deployment** profile to behave as shown in the lifecycle path below, you would apply a policy that enables the **Deployment** profile when the virtual service switches to the **VS_Virtualized**, **VS_Certified** or **VS_Promoted** state, and you would apply a policy to disable the **Deployment** profile when the virtual service switches to the **VS_Revising** or **VS_New** state.

Virtual Service Lifecycle Path with Deployment Status



Creating a Policy that Enables the Deployment Profile

To enable the **Deployment** profile for a virtual service, create a policy that contains the following action and apply this policy (on a PostStateChange) to all states during which you want the controls on the **Deployment** profile to be enabled.

```
Change Deployment Status (enable)
 ↩
```

If you want to prevent users from modifying the processing steps for a virtual service after the **Deployment** profile is enabled, include the Processing Step Status action in the policy to disable the **Processing Step** profile as shown below.

```
Processing Step Status (disable)
Change Deployment Status (enable)  ↩
```

Creating a Policy that Disables the Deployment Profile

To disable the **Deployment** profile, create a policy that contains the following action and apply this policy (on a PostStateChange) to all states during which you want the controls on the **Deployment** profile to be disabled.

```
Change Deployment Status (disable)
 ↩
```

If your lifecycle includes policies that automatically disable the **Processing Step** profile when the **Deployment** profile is enabled, this policy should include the Processing Step Status action to re-enable the **Processing Step** profile as shown below.

```
Change Deployment Status (disable)
Processing Step Status (enable)  ↩
```

Note: To see how these policies are implemented in an actual lifecycle model, install the starter kit and examine the policies associated with the lifecycle model for services and virtual services. For more information about the starter kit, see ***Obtaining and Installing the Starter Kit***.

Creating Virtual Services

To create a virtual service in CentraSite, you must select the native service for which you want to create the virtual service and run the “Virtualize” command. During the virtualization process, CentraSite copies the metadata (including the WSDL) from the native service to the virtual service. In other words, the virtual service is basically cloned from the native service.

Because the virtual service is cloned from the native service, it has its own copy of the service metadata. If you make a change to the metadata in the native service after you generate the virtual service, you will need to explicitly update the virtual service if you want that change reflected in the virtual service, too.

Note: If the native service includes file attributes that refer to documents in the supporting document library, the virtual service will reference the same documents. CentraSite does not create separate copies of the supporting documents for the virtual service. The virtual service simply refers to the same supporting documents as the native service.

When Should You Create a Virtual Service?

Generally speaking, you do not want to generate the virtual service unless the following conditions are satisfied:

- The interface for the native service is completely implemented and that interface is reflected in the WSDL that is registered for the service in CentraSite.
- An instance of the native service is deployed and running at a known point in network.
- The metadata for the native service is valid and up-to-date. If the metadata for the native service has not been completely specified or is out-of-date, you should update it before you generate the virtual service so that you do not carry inaccurate/incomplete data into the virtual service.

Important: Take care when assigning names to your virtual services. The name given to a virtual service when it is created, cannot be changed afterwards.

Who Should Create a Virtual Service?

If a user has View permission on a native service and "Create Assets" permission within their own organization, he or she can create a virtual service. However, the user will not be permitted to configure the processing steps for the virtual service unless he or she also has the "Manage Runtime Policies" permission for their organization. Only users with "Manage Runtime Policies" permission can configure these steps.

Consider identifying a small group of users who will be responsible for configuring the processing steps for a virtual service. Give this group a role that includes the "Manage Run-time Policies" permission. Because these users might configure virtual services that other users have created, they will also need Modify permission on the virtual services. To ensure that these users can edit the virtual services that they need to configure, consider creating a design/change-time policy that automatically gives this group Modify permission on a virtual service when it is created.

Virtual Service Ownership

One issue to consider when creating virtual services is the issue of ownership. When you create a virtual service, CentraSite automatically adds the virtual service to *your organization* (even if the native service itself belongs to another organization). You cannot explicitly specify the organization to which you want the virtual service added.

The issue of ownership is important with respect to virtual services, because it determines which run-time policies are applied to the virtual service when it is deployed. If the native service belongs to another organization, the existing run-time policies for your organization might or might not be appropriate for it.

When you define the general process that your site will follow to create and deploy virtual services (i.e., when you determine who will create a virtual service, who will configure a virtual service, and who will deploy a virtual service), keep in mind that CentraSite always adds a virtual service to the organization of the user who creates it. Make sure that whatever process you adopt for creating virtual services places a virtual service in the appropriate organization.

Deploying a Virtual Service

There are several ways you can deploy a virtual service to a Mediator instance. All methods except the first one allow you to deploy multiple virtual services in a single step.

- From the virtual service's detail page.
- From the **Operations > Deployment** page.
- From the target's detail page.
- Running a script file from a command line.
- Running a batch file.

To deploy a virtual service, the following conditions must be satisfied:

- Ensure that you have the "Manage RuntimeTargets" permission. Only users with this permission can deploy a virtual service. CentraSite will not enable the deployment controls for any other users.
- Ensure that the run-time policies for the virtualized service are active. This is indicated in the **Policies** profile on the virtualized service's detail page. If a policy is inactive, you must activate it as described in the section *Virtual Services in CentraSite Control* > *Creating Run-Time Policies* .

- Ensure that the virtualized service has a design-time policy that includes the Change Deployment Status action and it is set to Yes. This action specifies whether the service is eligible for deployment. For more information about this action, see the section *Built-In Design/Change-Time Actions Reference > Built-In Actions for Design/Change-Time Policies* .
- Ensure that the virtualized service has at least one target associated with it, and the target has must already have been created, as described in the section *Run-Time Targets*.
- Ensure that the target's specified deployment URL is active and the user credentials of Integration Server are valid. To check this, go to the target's detail page and click the **Check Connection** button. If the connection is not active and valid, activate the deployment endpoint and modify the user credentials as required.
- Ensure that the virtualized service is in a “deployable” lifecycle state. If you are not certain in which lifecycle states a virtualized service is eligible for deployment, consult your CentraSite administrator.

If these conditions are not satisfied, all or part of the deployment user interface controls will be disabled when you view the virtual service.

Note: Only users that are completely familiar with your site's mediation environment should be given permission to deploy virtual services. Generally, this would include a small number of administrators who have operational responsibility for the Mediators on which virtual services are deployed.

The Deployment Process

The deployment process is carried out by a sequence of interactions that occur between CentraSite and the Mediator:

1. CentraSite pushes the virtualized service that is ready for deployment to the webMethods Mediator target.
2. Instantly, the Mediator deploys the virtualized service that was received from CentraSite (along with its effective run-time policy), and notifies CentraSite when the deployment process is complete.

Undeploying a Virtual Service

After you deploy a virtual service to a Mediator, the virtual service remains deployed and active on that Mediator until you manually undeploy. You can deploy a virtual service using the same deployment mechanisms mentioned above.

Redeploying a Virtual Service

A virtual service that is already deployed on a Mediator can be manually redeployed. You can redeploy a virtual service using the same deployment mechanisms mentioned above. If you make changes to a virtual service's processing steps, for example, you must manually redeploy the virtual service to put those changes into effect.

As described in **Modifying a Run-Time Policy**, you cannot make changes to a run-time policy while it is active. To make changes to a policy after it has been switched to the active state you must do one of the following:

1. Switch the policy to the Suspended state (to deactivate it), update the policy and then switch it back to the Productive state (to reactivate it).

   OR

2. Create a new version of the policy, make your changes to the new version of the policy and then switch the new version to the Productive state. Switching the new version of the policy to the Productive state will automatically Retire (and deactivate) the old version.

If you need to update a run-time policy that is already deployed with virtual services that are in production, always use the second method described above (i.e., create a new version of the policy). If you use the first method, which requires you to suspend the existing policy, your production services will be running without the policy while you are making your revisions to it.

Revising a Virtual Service

Web services are bound to change and evolve over time. The loose coupling principles of service-oriented architecture (SOA) imply that service providers can release a new version of a shared service without waiting for consumers to adapt, and that service consumers should test and certify on a new shared service version before switching. Consequently, you might need to have multiple versions of a shared service running concurrently and simultaneously accessible by different service consumers. Some service consumers might need to continue using an old version of a service until migration of the consumer code occurs. Therefore, Web services versioning is an important subject that should be considered carefully in all enterprise SOA approaches.

Current standards for Web services have no explicit support for versioning. However, there are two alternatives for handling access to multiple versions of Web services:

1. Require the consumer applications to change their code to specify which versions to access.

   This option is rarely implemented due to its prohibitively complex and time-consuming nature.

2. Use a mediation layer (e.g., Mediator) to decouple the consumer from the provider, and thus allow the mediation layer to route requests to the desired version of a given service.

Mediator provides versioning solutions that you can implement, called “versioning patterns”. To implement versioning patterns, you configure virtual services in CentraSite so that consumers can access the desired version of a given service. You can use the versioning patterns to handle access to both Minor and Major versions of services.

Mediator cannot run multiple versions of the same virtual service simultaneously. Mediator only retains the last deployed version of a virtual service. However, suppose you have multiple versions of a native Web service. By using a versioning pattern, a single virtual service can provide access to the various native service versions based on an intelligent routing scheme that routes requests from a particular consumer to the correct native service version. A second option would be to provide multiple virtual services that correspond to multiple native service versions. For example, suppose you have two versions of the native service “GetOrder”. You have the following options in Mediator:

- Provide a single virtual service that intelligently routes each consumer to the appropriate “GetOrder” version (either version 1 or version 2).

  Or

- Provide one virtual service that routes consumers to version 1, and one virtual service that routes consumers to version 2.

This section discusses the following topics:

Minor Versions vs. Major Versions

Minor and Major versions of Web services are described as follows:

- **Minor version:**

  A Minor version is a version that is compatible with all consumers of the existing virtual service. That is, the changes in a Minor version do not “break” the existing applications that use the service. Examples of changes for a Minor version include:

  - Bug fixes.
  - Performance improvements.
  - The addition of a supporting document.
  - The addition of operations (as long as it does not break the existing applications).
  - A change in the Description attribute.

- **Major version:**

  A Major version is a version that is *incompatible* with consumers of the existing virtual service. That is, the changes in a Major version "break" the existing applications that use the service. Examples of a Major version include:

  - Modifications to the namespace assignments.
  - Modifications to message descriptions.
  - Modifications to interface definitions and/or operation signatures in the service WSDL.
  - Changes to the implementation of the service that do not explicitly affect the WSDL, but nevertheless affect the way in which an existing consumer application interacts with the service.

    For example, a service that returns an expanded set of result codes or generates a different form of customer ID might break an existing consumer application even if the interface defined in the service WSDL did not change.

Note: Be aware that sometimes versioning one asset will necessitate the versioning of another. For example, if an XML schema changes, and that schema is imported by a Web service, you will need to generate a new version of the XML schema and a new version of the Web service that references it.

The Layer of Indirection Pattern

This pattern allows multiple Minor versions of a native service to coexist in the registry without requiring consumers to change the code in their consumer applications, and helps to ensure a graceful migration to the new Minor version.

To implement this pattern, you configure a single virtual service to route each request to the version that is appropriate for each consumer (or group of consumers). That is, you configure the virtual service's **Routing step** to use either the "content-based routing" option or the "context-based routing" option.

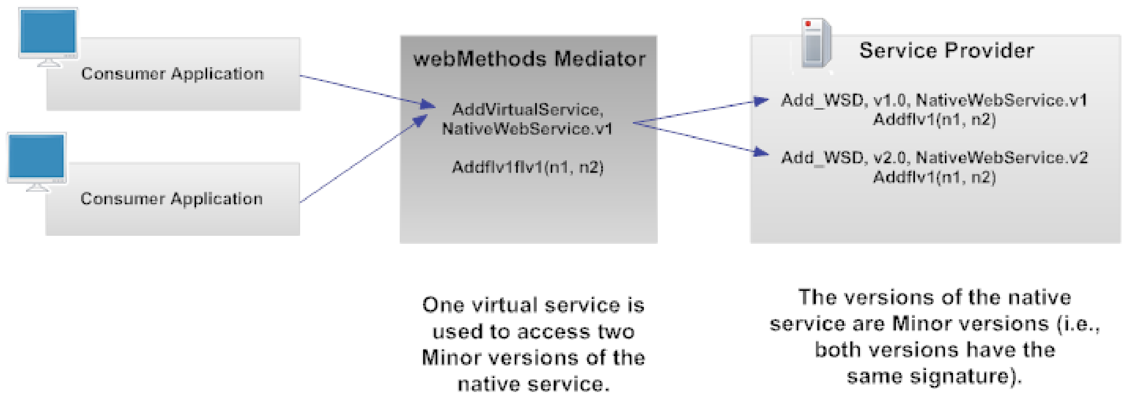
- **Content-based routing option:**

  Using the content-based routing option, you can route request messages to different endpoints based on specific values that appear in the request message. For example, if a new Minor version contains an additional operation, you can write a rule that routes all requests that reference the newly-added operation to the new Minor version.

- **Context-based routing option:**

  Using the context-based routing option, you can route request messages to different endpoints based on the identity of the consuming application. For example, if you want to allow only certain consumers to access a new Minor version, you write a rule that routes only their requests to the new Minor version.

The Layer of Indirection Pattern


Consumer Application
Consumer Application
webMethods Mediator
AddVirtualService, NativeWebService.v1
Addflv1flv1(n1, n2)
Service Provider
Add_WSD, v1.0, NativeWebService.v1
Addflv1(n1, n2)
Add_WSD, v2.0, NativeWebService.v2
Addflv1(n1, n2)
One virtual service is used to access two Minor versions of the native service.
The versions of the native service are Minor versions (i.e., both versions have the same signature).

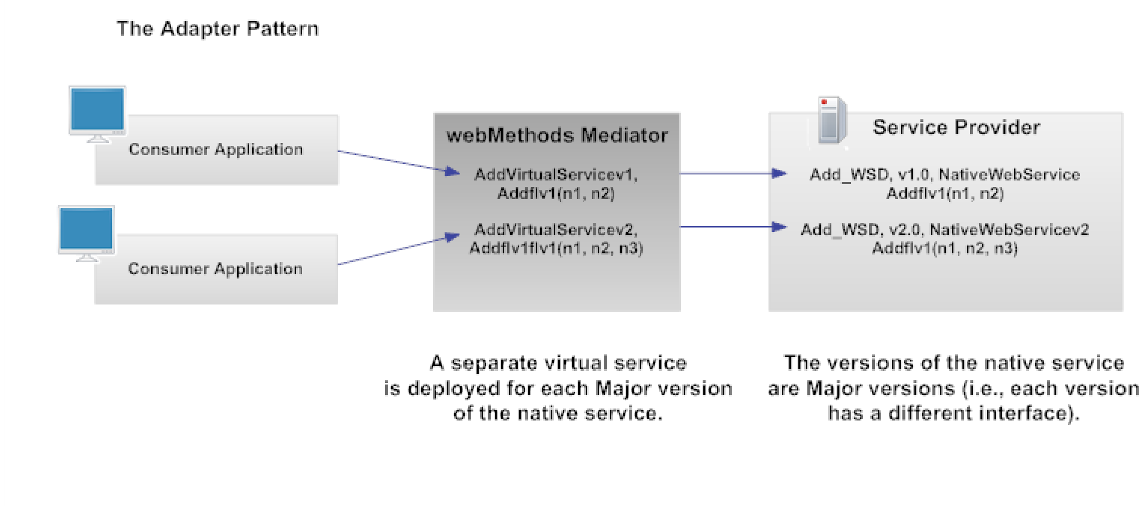
The Adapter Pattern

This pattern allows multiple *Major* versions of a native service to coexist in the registry without requiring consumers to change the code in their consumer applications, and helps to ensure a graceful migration to the new Major version.

This is called the "adapter pattern" because Mediator will act as an adapter, adapting the client requests before they are submitted to the native services.

Unlike the Layer of Indirection pattern -- which has *one* virtual service that can access each Minor version of the native service -- this pattern has *a separate virtual service for each Major version of the native service.*

To implement this pattern, you configure the virtual service's **Request Processing step** so that it transforms the endpoint specified in a request to the endpoint of the desired version. The Request Processing step specifies how the request message is to be transformed before it is submitted to the native service. You can configure this step to perform message transformations using a specified XSLT file or by passing the message to a webMethods IS service (i.e., an Integration Server service running on the same Integration Server as webMethods Mediator).


The Adapter Pattern
Consumer Application
Consumer Application
webMethods Mediator
AddVirtualServicev1, Addflv1(n1, n2)
AddVirtualServicev2, Addflv1flv1(n1, n2, n3)
Service Provider
Add_WSD, v1.0, NativeWebService Addflv1(n1, n2)
Add_WSD, v2.0, NativeWebServicev2 Addflv1(n1, n2, n3)
A separate virtual service is deployed for each Major version of the native service.
The versions of the native service are Major versions (i.e., each version has a different interface).

Combination of the Layer of Indirection Pattern and the Adapter Patterns

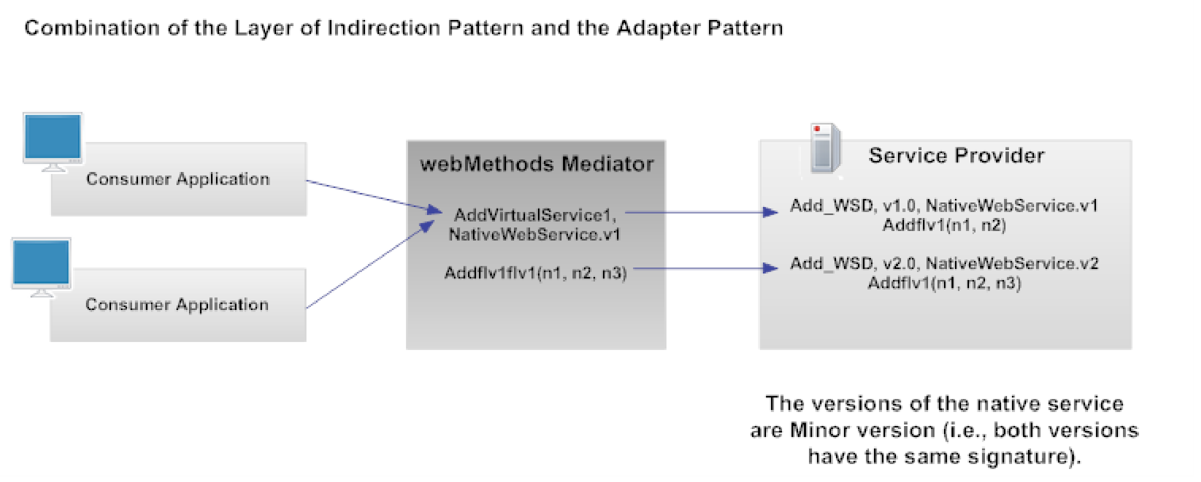
This pattern has one virtual service that can access:

- Multiple Major versions of a native service (i.e., utilizing the Adapter pattern).

  Thus you configure the virtual service's Request Processing step so that it transforms the endpoint specified in a request to the endpoint of the desired version..

- Multiple Minor versions of a native service (i.e., utilizing the Layer of Indirection pattern).

  Thus you configure the virtual service with either the "content-based routing" option or the "context-based routing" option in order to route requests to the Minor version that is appropriate for each consumer (or group of consumers).


Combination of the Layer of Indirection Pattern and the Adapter Pattern
Consumer Application
Consumer Application
webMethods Mediator
AddVirtualService1, NativeWebService.v1
Addflv1flv1(n1, n2, n3)
Service Provider
Add_WSD, v1.0, NativeWebService.v1 Addflv1(n1, n2)
Add_WSD, v2.0, NativeWebService.v2 Addflv1(n1, n2, n3)
The versions of the native service are Minor version (i.e., both versions have the same signature).

16 Identifying the Consumers of Virtual Services

In CentraSite there are two concepts of “consumers”.

- The first refers to developers who discover assets in the catalog that they want to reuse. Such developers can register to become *registered consumers* of those assets. You might give registered consumers access to more of the asset's metadata (i.e., enable them to view additional profiles) and/or develop processes that notify them when changes occur to an asset that they consume.
- The second concept refers specifically to a computer application that consumes (invokes) virtual services at run time. This specific type of consumer is represented in the registry by instances of the *Application* asset type. Application assets are used by webMethods Mediator to determine from which computer application a request for a virtual service originated.

  This section discusses the type of consumer that represents a computer application that consumes virtual services. For clarity, the following sections in this document refer to this type of consumer as a *consumer application*.

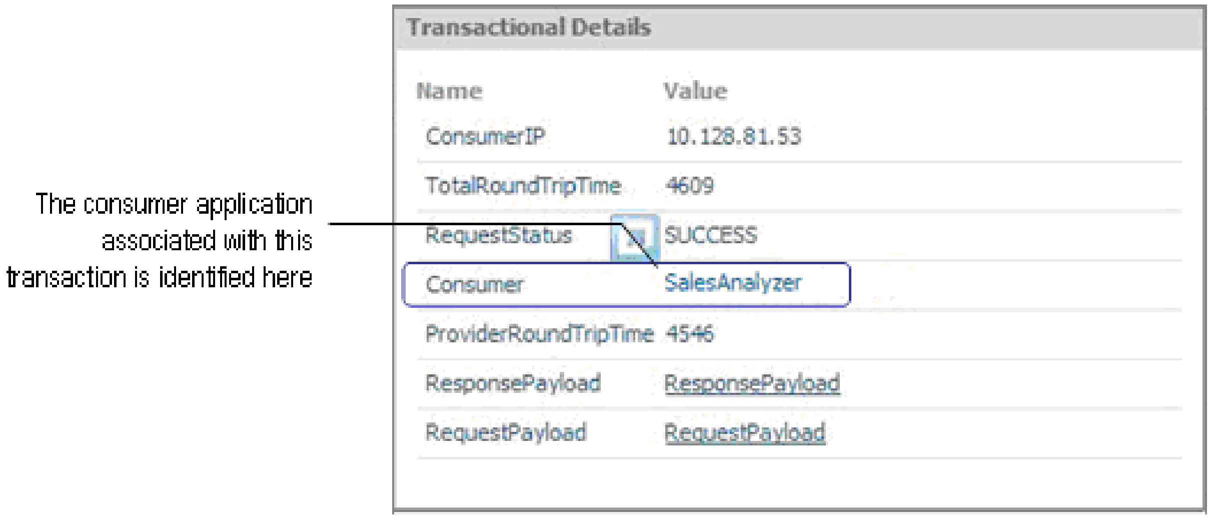
How Are Consumer Applications Represented and Used in CentraSite?

A consumer application is represented in CentraSite by an *application asset*. An application asset is an instance of the Application asset type, which is one of the predefined types installed with CentraSite. An application asset defines the precise characteristics by which Mediator can identify messages from a specific consumer application at run time.

The ability of Mediator to relate a message to a specific consumer application enables Mediator to:

- Indicate the consumer application to which a logged transaction event belongs.
- Monitor a virtual service for violations of a service-level agreement (SLA) for a specified consumer application.
- Control access to a virtual service at run time (i.e., allow only authorized consumer applications to invoke a virtual service).

The following figure shows the log entry for a request that a consumer application has submitted to a virtual service. Note that the entry identifies the consumer application from which the request originated. This identification is enabled by an application asset that has been defined in the CentraSite registry.


The consumer application associated with this transaction is identified here
Transactional Details

Name	Value
ConsumerIP	10.128.81.53
TotalRoundTripTime	4609
RequestStatus	SUCCESS
Consumer	SalesAnalyzer
ProviderRoundTripTime	4546
ResponsePayload	ResponsePayload
RequestPayload	RequestPayload

How Does Mediator Identify Consumer Applications at Run Time?

To determine the consumer application from which a request was submitted, a virtual service must have a run-time policy that includes the Identify Consumer action. This action extracts a specified identifier from an incoming request and locates the application asset defined by that identifier.

For example, if you configure the Identify Consumer action to identify consumers by IP address, Mediator extracts the IP address from a request's HTTP header and searches its list of application assets for the application that is defined by that IP address.

You can configure the Identify Consumer action to identify consumer applications based on the following information in a request message.

Identifier	Description
IP Address	The IP address from which the request originated.
Host Name	The name of the host machine from which the request originated.
HTTP Authentication Token	The user ID submitted by the requestor when it was asked to provide basic HTTP credentials (user name and password).
WS-Security Authentication Token	The WSS username token supplied in the header of the SOAP request that the consumer application submitted to the virtual service.
Consumer Certificate	The X.509 certificate supplied in the header of the SOAP request that the consumer application submitted to the virtual service.

Defining Application Assets in CentraSite

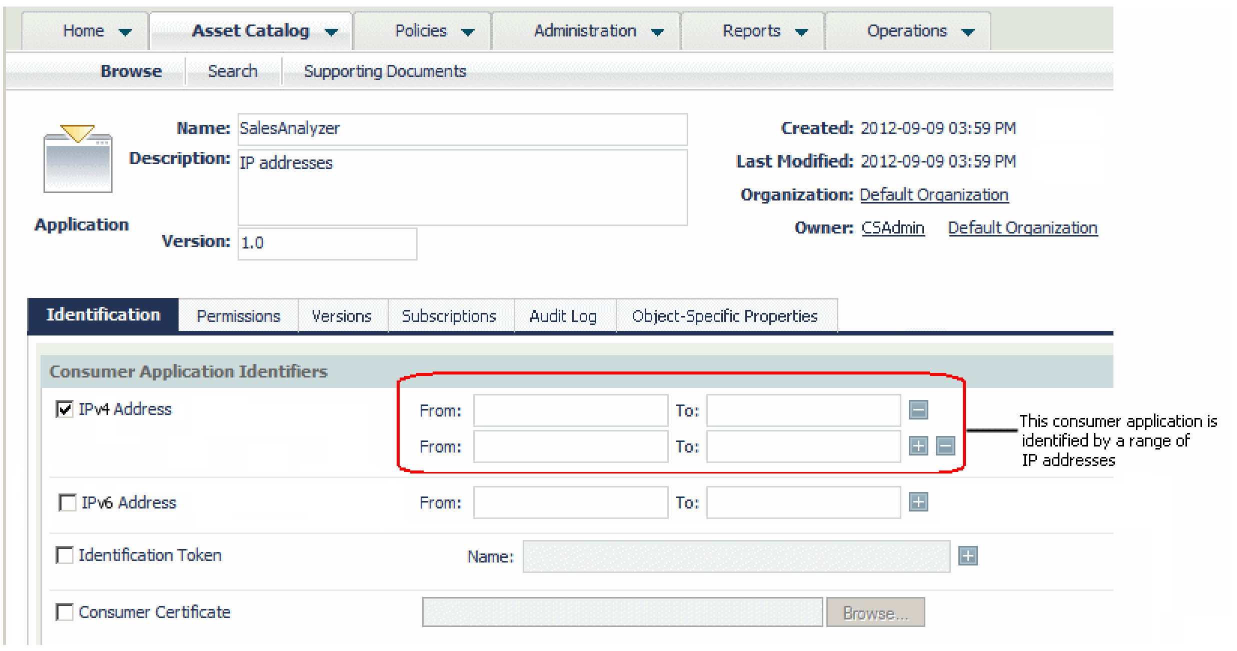
An application asset specifies the precise identifiers by which messages from a particular consumer application will be recognized at run time. An application asset has the following attributes for specifying these identifiers:

- *IPv4 Address*, which specifies one or more 4-byte IPv4 addresses that identify requests from a particular consumer application. (This attribute is queried when the Identify Consumer action is configured to identify consumer applications by IP address.)

  Example: 192.168.0.10
- *IPv6 Address*, which specifies one or more 128-bit IPv6 addresses that identify requests from a particular consumer application. See the IPv6 addressing architecture specification **ht-tp://tools.ietf.org/html/rfc4291** for details of this format.

  Example: 1234:5678:9ABC:DEF0:1234:5678:9ABC:DEF0
- *Identification Token*, which specifies the host names, user names or other distinguishing strings that identify requests from a particular consumer application. (This attribute is queried when the Identify Consumer action is configured to identify consumer applications by host name, HTTP user name, WSS user name or a custom token.)
- *Consumer Certificate*, which specifies the X.509 certificates that identify requests from a particular consumer. (This attribute is queried when the Identify Consumer action is configured to identify consumer applications by a consumer certificate.)

For example, the following application asset describes a consumer application called SalesAnalyzer, which is defined by a range of IP addresses.



Name: SalesAnalyzer
Description: IP addresses
Application
Version: 1.0
Created: 2012-09-09 03:59 PM
Last Modified: 2012-09-09 03:59 PM
Organization: Default Organization
Owner: CSAdmin Default Organization

Consumer Application Identifiers

☑ IPv4 Address From: To: From: To:
☐ IPv6 Address From: To:
☐ Identification Token Name:
☐ Consumer Certificate

This consumer application is identified by a range of IP addresses

Synchronizing the Application Assets in CentraSite with the Mediator

When Mediator identifies a consumer application at run time, it searches a local list of application assets that it maintains. This list is initially downloaded from the CentraSite registry when you start Mediator. Mediator periodically resyncs the list to keep it up-to-date.

Be aware that application assets are made available to Mediator as soon as they are added to the registry. That is, an application asset that you add to CentraSite is given to Mediator the next time Mediator resyncs its local list with the registry. Therefore, you should not add an application asset to CentraSite's registry until you are able to provide the proper identifiers for it.

Deciding How to Identify a Consumer Application

When deciding which type of identifier to use to identify a consumer application at run time, consider the following points:

- Whatever identifier you choose to identify a consumer application, it must be unique to the application. Identifiers that represent user names are often not suitable because the identified users might submit requests from multiple consumer applications.
- Although identifying applications by IP address or host name is often a suitable choice, it does create a dependency on the network infrastructure. If a consumer application moves to a new machine, or its IP address changes, you must update the identifiers in the application asset.
- Using X.509 certificates or a custom token that is extracted from the SOAP message itself (using an XPATH expression), is often the most trouble-free way to identify a consumer application.

Note: Depending on which form of identification you choose, the run-time policy that you use to extract the consumer identifier might need to perform certain prerequisite actions prior to the Identify Consumer action. For example, if you want to identify a consumer application by WS-Security authentication token, your run-time policy must execute the Require WSS Username Token action *before* it executes the Identify Consumer action. These dependencies are described in the user documentation for the Identify Consumer action.

Registering an Application Asset with a Virtual Service

You use the **Register as Consumer** command in CentraSite to associate an application asset with a virtual service. This command establishes an association between the application asset (which represents a consumer application) and the virtual service that it consumes. Registering an application asset with a virtual service enables you to use the Impact Analysis feature in CentraSite to quickly determine which virtual services a consumer application consumes (and also determine which consumer applications use a particular virtual service).

Additionally, if you use the Authorize Against Registered Consumers policy action to control access to a virtual service at run time, only registered consumer applications are allowed to invoke the virtual service. Consequently, when you use this form of access control, the consumer applications that are permitted to use a virtual service *must be registered to the virtual service*.

The Consumer Registration Process

When you execute the Register as Consumer command, CentraSite does not immediately register the application asset with the virtual service. Instead, it triggers a review and approval process that includes the following steps:

1. CentraSite submits the request to the owner of the virtual service for review and approval.
2. The virtual service owner reviews the request and approves the request or rejects it. If the request is approved, CentraSite executes the consumer-registration policy. This policy actually registers the application asset with the virtual service.

The Consumer-Registration Policy

The consumer-registration policy is a policy that includes the Register Consumer action and executes on the OnConsumerRegistration event. The OnConsumerRegistration event occurs *when the owner of the asset approves the registration request*, not when the user submits the registration request. *CentraSite does not provide a consumer-registration policy out-of-the-box*. You must create this policy for your instance of CentraSite. CentraSite will not enable the consumer-registration feature until you create this policy.

Important: If you will be using the Authorize Against Registered Consumers policy action to control access to a virtual service at run time, you should strongly consider including an approval step in your consumer-registration policy. When you use this form of access control on a virtual service, registering a consumer application with the virtual service grants that consumer application permission to invoke the service. To ensure that only authorized applications are registered with a virtual service, you might want to have a security administrator review and approve this type of registration request.

Issues to Consider when Defining Applications

When defining application assets, keep the following points in mind:

- Any user who has permission to publish an asset to CentraSite can define an application asset. However, not all users are generally qualified to create an asset of this type. Defining applications is a critical task that should be performed only by an administrator who is familiar with the webMethods Mediator(s), virtual services and run-time policies in your environment.
- An application asset becomes available to Mediator as soon as you add it to the registry. Do not add an application asset to CentraSite until you are able to provide the exact identifiers for the consumer application that it represents.
- Treat application assets as global objects and make them available to all organizations. Be sure that your registry contains only one application asset per consumer application (i.e., a consumer application should be represented by *one and only one application asset* in the registry).
- Be sure that the identifiers that you assign to an application asset are unique to that application asset. If multiple application assets have the same identifier, Mediator will simply associate the identifier with the first matching application it finds in its local list of application assets at run time.
- If you control access to virtual services based on consumer applications (i.e., you use run-time policies that include the Authorize Against Registered Consumers action), consider:
  - Including an approval step in your consumer-registration policy that requires a security administrator to review and approve the registration event.
  - Giving only a small group of knowledgeable administrators permission to modify an application asset after it is registered to a virtual service. This will prevent users from adding unauthorized identifiers to an existing application asset, and thus, allowing unauthorized consumer applications to access the virtual service.

17 Managing Endpoints

In an SOA environment, the tracking and management of service endpoints is a key task. Not only is your SOA environment likely have many individual services, but many of those services will be deployed on multiple endpoints.

In general, a service will have multiple endpoints for the following reasons:

- It acquires additional endpoints as it moves through its lifecycle. For example, by the time a service goes into production, it is usually available at three different endpoints in your environment: at a development endpoint, at a test endpoint and at a production endpoint. To track the endpoints that a service acquires as it moves through its development cycle, you add the endpoints to the metadata for the service in the registry.
- It has different endpoints to accommodate the needs of different consumer applications. For example, you might offer the same service over multiple entry protocols (e.g., both JMS and HTTP) and/or with different security mechanisms. In CentraSite, you accommodate these needs by exposing the service over multiple virtual services.

Who Uses Endpoint Information?

Endpoint information is needed by developers who write programs that bind to services.

At run time, a consumer application can bind to a service in one of two ways: it can perform a static bind or a dynamic bind.

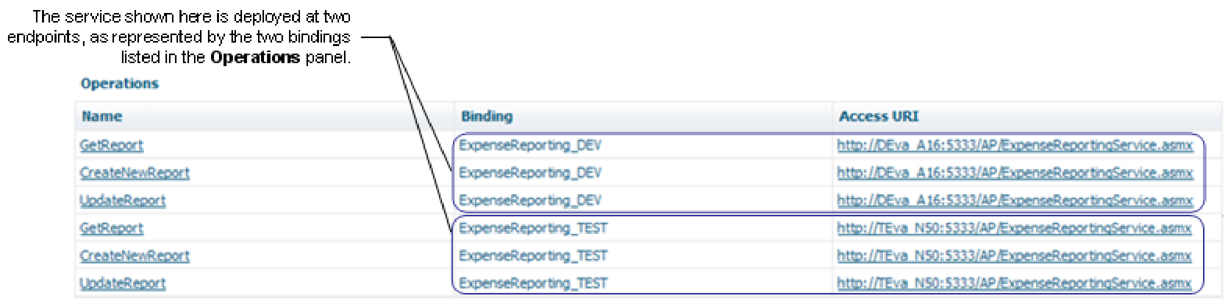
- When a consumer application is written to use static binding, the program binds to a specified address at run time (i.e., an endpoint that is already known by the program). When a developer creates a program that uses static binding, he or she will get the endpoint of the service from CentraSite at design time. Typically, this end point is added to a configuration file or a parameter setting that the consumer program reads at run time. If the actual endpoint of the service ever changes, the consumer program must be configured to access the service at its new endpoint.
- When a consumer application is written to use dynamic binding, the program “looks up” the service's address at run time and binds to that address. To use this type of binding, a developer must write a query to retrieve the service's endpoint from CentraSite. At run time, the consumer program executes the query and binds to the endpoint that the query returns. If the endpoint of the service changes, one only has to update the service's endpoint information in CentraSite. Nothing has to be changed in the consumer application.

The endpoint information that you maintain in CentraSite can be used for either purpose.

How Service Endpoints Are Represented in CentraSite

Within CentraSite, the endpoints for a service are shown in the **Operations** panel on the service's **Summary** profile. In this panel, an endpoint is represented as a **Binding** that identifies a specific **Access URI** (i.e., address where the service is deployed).

The following example shows the **Operations** panel for a service that is deployed at two endpoints: the endpoint represented by the **ExpenseReporting_DEV** binding and the endpoint represented by the **ExpenseReporting_TEST** binding. Note that the **Access URI** column provides the exact address of each endpoint.

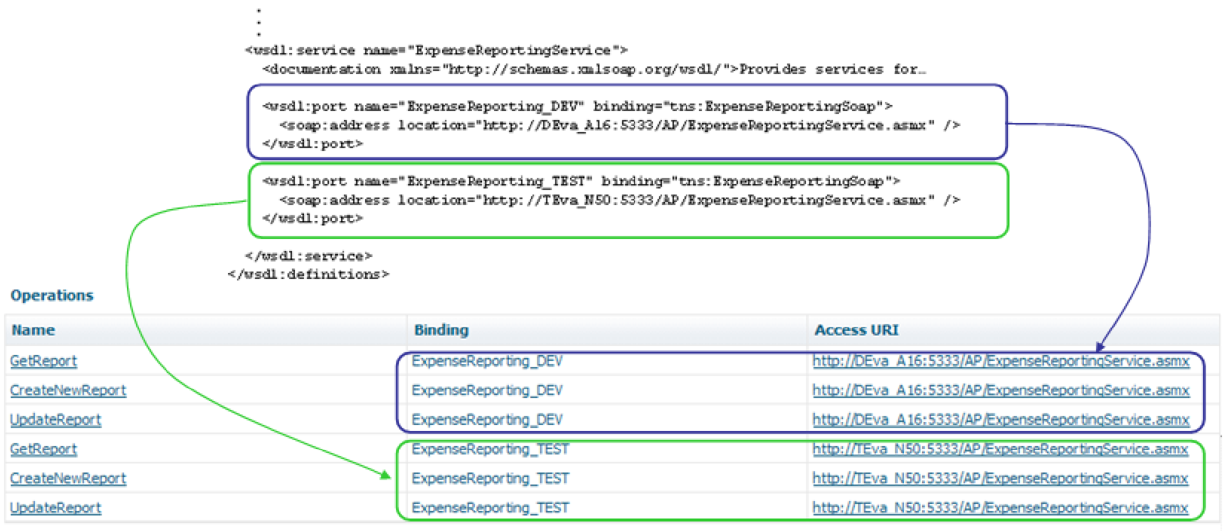

The service shown here is deployed at two endpoints, as represented by the two bindings listed in the **Operations** panel.

Operations

Name	Binding	Access URI
GetReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
GetReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx

The **Binding** and **Access URI** information that appears in the **Operations** panel is derived from the <port> definitions in the service WSDL. Specifically, the **Binding** name is derived from the name of the port and the **Access URI** is derived from the port's <address> element.

The Bindings on the Operations panel are derived from the <port> definitions in the service WSDL



```
 :
 :
<wsdl:service name="ExpenseReportingService">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Provides services for…

  <wsdl:port name="ExpenseReporting_DEV" binding="tns:ExpenseReportingSoap">
    <soap:address location="http://DEva_A16:5333/AP/ExpenseReportingService.asmx" />
  </wsdl:port>

  <wsdl:port name="ExpenseReporting_TEST" binding="tns:ExpenseReportingSoap">
    <soap:address location="http://TEva_N50:5333/AP/ExpenseReportingService.asmx" />
  </wsdl:port>

</wsdl:service>
</wsdl:definitions>
```

Operations

Name	Binding	Access URI
GetReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
GetReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx

Managing the Endpoints of a Native Service over its Lifecycle

When a native service moves through its lifecycle, it usually gains additional endpoints. For example, during development, a developer generally deploys the service somewhere in the development environment. When the service moves to the testing phase, it is generally deployed at another endpoint for testing. Finally, when the service is placed in production, the operations organization deploys the service at an endpoint in the production environment.

Each time you deploy a native service to an additional endpoint, you must add the new endpoint to the service in CentraSite. To do this you:

1. Download the service WSDL from the CentraSite registry.
2. Add the endpoint to the WSDL (as an additional port definition).
3. Reattach the updated WSDL to the service in CentraSite.

When you attach the updated WSDL to the service, CentraSite will automatically update the binding information on the service's **Operations** panel.

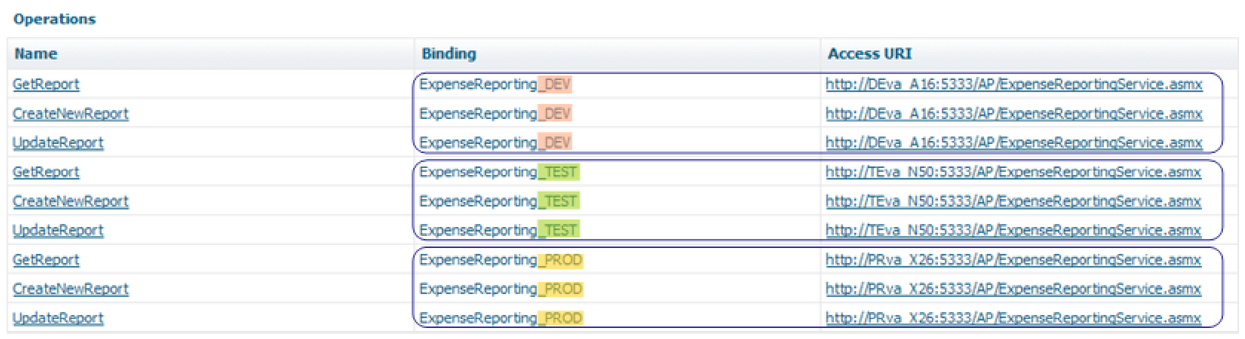
Note: Although it is possible to represent the development, test and production endpoints as individual services in the registry, we recommend that you avoid doing this. Such an approach produces a large amount of duplicated metadata and does not return any real benefits. Instead, maintain just one catalog entry for a native service and add the service endpoints to this entry as the service progresses through its lifecycle.

Example of the Bindings for a Service in a Single-Stage Registry

If you are using a single-stage deployment of CentraSite, services remain in the same registry for their entire lifecycle. Therefore, in this type of registry, the catalog entry for a service will include the service's development, test and production endpoints.

The following shows an example of a service that has endpoints in the development, test and production environments. Note that the naming scheme that has been used to identify the bindings for this service clearly indicates the environment in which the endpoint resides.

In a single-stage environment, the development, test and production endpoints are listed for the service



Operations

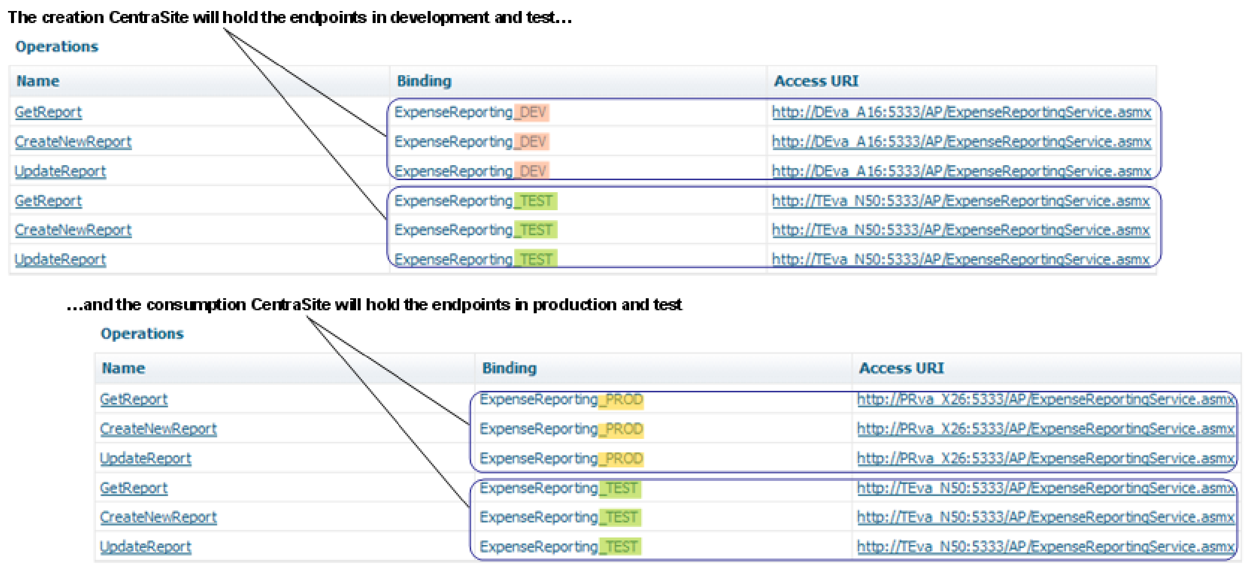
Name	Binding	Access URI
GetReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
GetReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
GetReport	ExpenseReporting_PROD	http://PRva_X26:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_PROD	http://PRva_X26:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_PROD	http://PRva_X26:5333/AP/ExpenseReportingService.asmx

Important: The endpoints for a service are visible to any user who has View permission on the service in CentraSite. To prevent unauthorized access to the services themselves, be sure that appropriate security measures are in place at these endpoints.

Example of the Endpoints for a Service in a Two-Stage Registry

In a multi-stage deployment, the set of endpoints that you publish to the registry will vary according to needs of the registry's audience. In a two-stage deployment, for example, you would list the service's development and test endpoints on the creation CentraSite and you would list the service's test and production endpoints on the consumption CentraSite.

The following shows an example of the bindings you would see if you were to view a service in the creation CentraSite and in the consumption CentraSite.



The creation CentraSite will hold the endpoints in development and test...

Operations

Name	Binding	Access URI
GetReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_DEV	http://DEva_A16:5333/AP/ExpenseReportingService.asmx
GetReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx

...and the consumption CentraSite will hold the endpoints in production and test

Operations

Name	Binding	Access URI
GetReport	ExpenseReporting_PROD	http://PRva_X26:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_PROD	http://PRva_X26:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_PROD	http://PRva_X26:5333/AP/ExpenseReportingService.asmx
GetReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
CreateNewReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx
UpdateReport	ExpenseReporting_TEST	http://TEva_N50:5333/AP/ExpenseReportingService.asmx

Adopting a Naming Convention for Binding Names

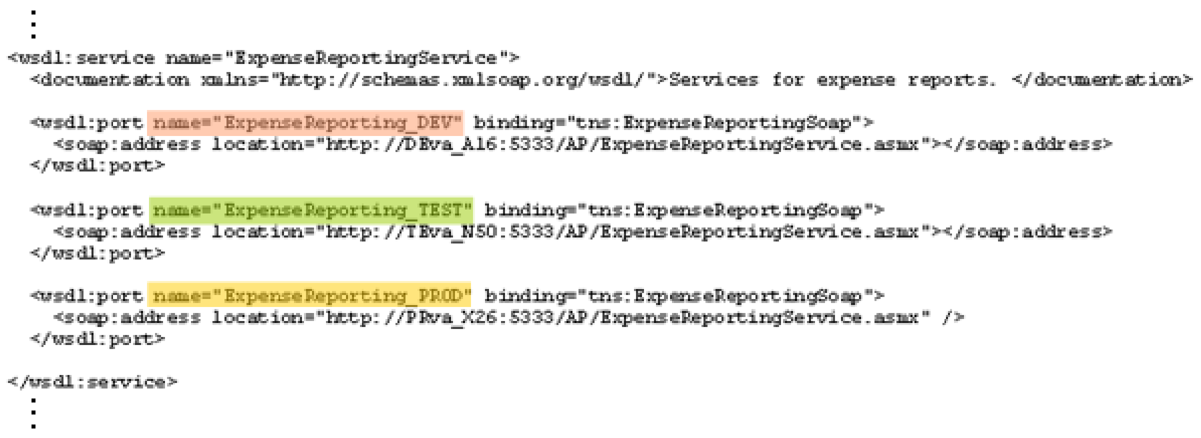
When a service has multiple endpoints, the binding names give users a hint as to the endpoint's function. As a best practice, consider adopting a naming convention for bindings that identifies service endpoints in a clear and consistent manner. (This practice is especially important if your consumer applications will be querying the registry to obtain a service endpoint run time.)

In the examples shown above, each binding name includes a suffix to indicate the environment in which the endpoint resides.

Keep in mind that binding names are derived directly from the port names in the service WSDL. Therefore, to produce bindings whose names conform to the particular naming scheme that you have adopted, you must assign the appropriate names to port definitions in the WSDL to begin with.

For example, to produce the binding names shown in the single-stage example described in **Example of the Bindings for a Service in a Single-Stage Registry**, the port definitions in the WSDL must look as follows.

Naming conventions for bindings must be applied to the port names in the WSDL



```
  :
<wsdl:service name="ExpenseReportingService">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Services for expense reports. </documentation>

  <wsdl:port name="ExpenseReporting_DEV" binding="tns:ExpenseReportingSoap">
    <soap:address location="http://DEva_A16:5333/AP/ExpenseReportingService.asmx"></soap:address>
  </wsdl:port>

  <wsdl:port name="ExpenseReporting_TEST" binding="tns:ExpenseReportingSoap">
    <soap:address location="http://TEva_N50:5333/AP/ExpenseReportingService.asmx"></soap:address>
  </wsdl:port>

  <wsdl:port name="ExpenseReporting_PROD" binding="tns:ExpenseReportingSoap">
    <soap:address location="http://PRva_X26:5333/AP/ExpenseReportingService.asmx" />
  </wsdl:port>

</wsdl:service>
  :
```

Managing Endpoints of a Virtual Service over its Lifecycle

When you create a virtual service, CentraSite generates a WSDL file for the virtual service. Initially, Mediator generates this WSDL file as an abstract WSDL file, and is represented as an "empty" WSDL in the format `<protocol>://`. However, when you deploy the virtual service, CentraSite replaces the port definitions in this WSDL file with a port definition that specifies the virtual service's endpoint on the Mediator. At this time, it also updates the binding information that appears on the virtual service's **Operations** panel.

CentraSite will automatically update the port definitions in the virtual service WSDL and regenerate the corresponding bindings any time you deploy, undeploy or redeploy the virtual service.

Because the endpoint information for virtual services is generated and updated by CentraSite, you should not manually edit the endpoint information for virtual services. In other words, unlike native services, you should not manually add endpoints to the WSDL of a virtual service. Instead, simply allow CentraSite to generate and manage the endpoints for the virtual services that you deploy.

Publishing the Test Endpoint for a Virtual Service on the Consumption Registry

In the consumption CentraSite, the catalog entry for a native service provides consumers with bindings to the test instance of the service and to the production instance of the service. However, with a *virtual service*, you cannot do this. The set of bindings for a virtual service are generated and managed automatically by CentraSite, and you cannot manually add bindings to this set.

If you have a test instance of a virtual service deployed on the Mediator in your test environment, and you would like to disclose that endpoint to users when they view the virtual service in the registry, you can identify the endpoint in a separate attribute (i.e., as additional metadata) within the virtual service on the consumption CentraSite.

Note: If you have consumer applications that will dynamically bind to a virtual service, be aware that those applications will need to bind against the *creation CentraSite* during the testing phase of their development and against the *consumption CentraSite* when they enter production.

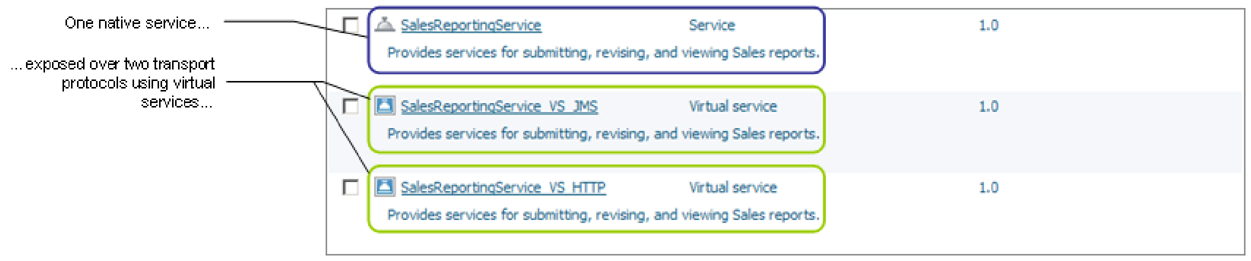
Deploying Multiple Virtual Services for a Single Native Service

Often you will need to deploy a service on multiple endpoints to make the service available over multiple transports and/or security mechanisms. For example, you might want to extend the same service over JMS and HTTP transports. Or, you might want to allow internal users to access a service using basic HTTP user name/password credentials and you might require other users to submit digital certificates.

To accommodate these kinds of operational requirements for a native service, you deploy multiple virtual services for a single native service. For example, to make a particular native service available to consumers over both HTTP and JMS, you would create two virtual services for the native service: one that accepts requests over HTTP and another that accepts requests over JMS. Both virtual services would route requests to the same native service on the back end.

The following shows a registry in which a native service (SalesReportingService) has been exposed over two virtual services.

Virtual Services provide two transports for one native service


One native service...
...exposed over two transport protocols using virtual services...

☐	SalesReportingService Provides services for submitting, revising, and viewing Sales reports.	Service	1.0
☐	SalesReportingService_VS_JMS Provides services for submitting, revising, and viewing Sales reports.	Virtual service	1.0
☐	SalesReportingService_VS_HTTP Provides services for submitting, revising, and viewing Sales reports.	Virtual service	1.0

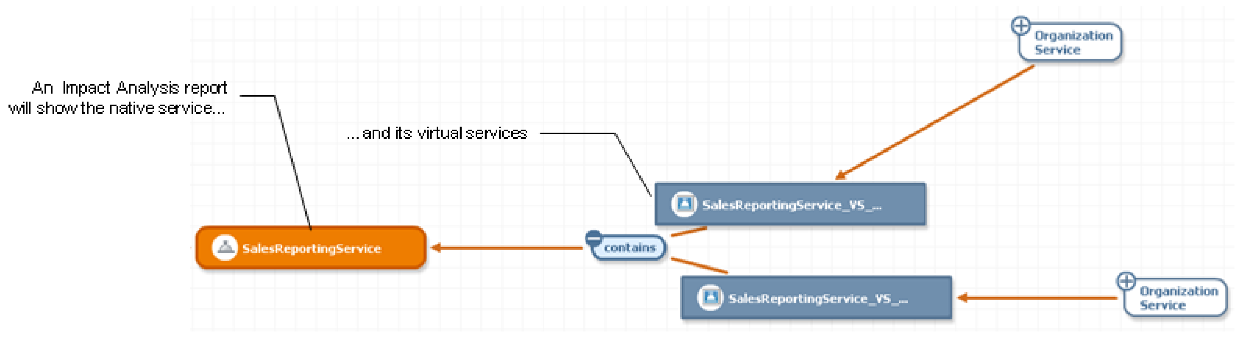
Note: To make it easier to manage virtual services, consider adopting a naming convention like the one shown above. Doing so will make it easier to identify virtual services and the native service with which they are associated. Keep in mind however, that unlike native services, the names of virtual services cannot contain spaces or special characters (except _ and -). Consequently, if you adopt a convention that involves using the name of the native service as part of the virtual service name, then the names of the native services themselves must not contain characters that are invalid in virtual service names.

Using the Impact Analysis Tool to Find the Virtual Services for a Native Service

When you create a virtual service, CentraSite establishes a relationship between the virtual service and the native service from which you created it. If you create multiple virtual services for a native service, each of the virtual services will have an established relationship to the native service.

The associations that CentraSite creates between a native service and a virtual service enables you to use the Impact Analysis tool to examine a native service and quickly locate all of its virtual services.

The Impact Analysis tool will list the virtual services associated with a native service


An Impact Analysis report will show the native service...
...and its virtual services
Organization Service
SalesReportingService_VS_...
SalesReportingService
contains
SalesReportingService_VS_...
Organization Service

Important: To ensure that a relationship is established between a native service and virtual service, always use the **Search for Endpoint** button to set the **Route To** address in a virtual service's **Routing Protocols** processing step. Do not manually type this address into the **Route To** field. If you type the address manually, the relationship to the appropriate native service will not be created.