BEA WebLogic Portal®

Communities Guide

Version 10.2
Revised: February 2008

Contents

3. Community Architecture

4. Planning and Designing Communities

Part II. Development

5. Developing Custom Communities

Part III. Staging

6. Setting up Community Infrastructure

Part IV. Production

7. Creating and Managing Communities

CHAPTER 1

Introduction to Communities

This chapter provides an overview of Community concepts and the Community framework. The other chapters in this guide describe the Community architecture and show you how to plan, build, deploy, and manage Communities.

This chapter includes the following sections:

Overview

Communities are WebLogic Portal desktops that let users with common goals and interests create, manage, and work in their own web-based environment. Communities provide a dedicated, secure, self-managed place for work groups, partners, or other groups to collaborate and share information.

Communities provide end-user Community management and an extensible security framework, and are built on the WebLogic Portal framework. Communities leverage WebLogic Portal features, such as a dynamic and extensible rendering framework, security, federation, content management, search, personalization, and end-user customization.

A Community is designed for a unique groups of users, so each Community contains unique information about itself. This information is called a Community context. The Community framework provides is an API that lets you access that Community context and develop

Community-specific functionality. For example, you could develop a portlet that lists the current members of a Community and provides a link to their contact information.

This section contains the following topics:

Community Examples

The following examples demonstrate ideal uses for Communities:

- An Internal HR Site – Each division within a company can create and manage their own human resources Community. The Community can contain global HR portlets and portlets that are specific to each division, such as organizational charts and contact information.
- A Franchise – Each affiliate can create and manage its own Community desktop. The Community can simply act as the affiliate web site that is managed by the affiliate.
- A Brokerage – The brokerage can create a Community desktop for each commodity being traded. In each Community, members view commodity-specific news and data, see trading alerts, perform transactions, and participate in discussion groups. The brokerage could also create a Community that lets members customize their own Community views. For example, members can add portlets for commodities they want to track.
- A Software Company – The company can create internal team or project management Communities. Each project team can have its own self-managed Community where members can share documents, manage tasks and issues, track progress, and communicate with each other. WebLogic Portal's GroupSpace Community template is an ideal solution for this type of Community. For more information see the *GroupSpace Guide*.
- An Online Music store – The store can create a Community for each music genre. Members can view record information, learn about new artists in the genre, read artist biographies, see tour dates, participate in discussion groups, and buy music.

When to Use Communities

Communities are portal desktops. They include a collection of books, pages, and portlets, have specific features and functionality, and have unique look and feels; but they provide additional features.

Use Communities when any of the following apply:

- Users need to be able to create and manage their own Community desktops
- Groups of users need a collaborative environment with application functionality that is unique to that collaborative environment
- You want to create custom membership roles that can affect application functionality in unique ways

If you want to maintain complete, centralized management control over desktops, or if there is no need for collaboration, create regular portal desktops rather than Communities. See the *Portal Development Guide* for instructions.

Managing Communities

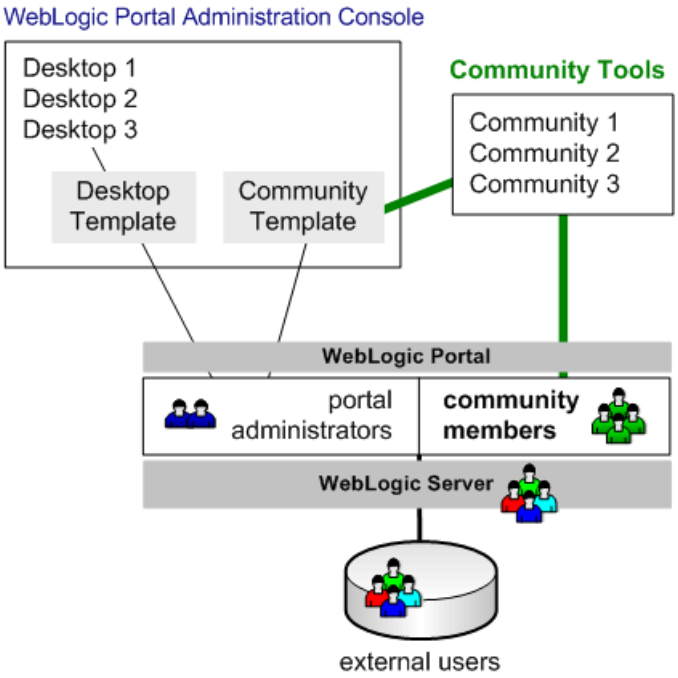
One of the most important characteristics of Communities is that members can create and manage their own Communities without relying on centralized portal administrators, who use the WebLogic Portal Administration Console. WebLogic Portal's Community Tools are available as menu choices within the Community.

Authorized members can perform the following types of administrative tasks:

- Send invitations to people to become Community members, or rescind invitations
- Set access capabilities on Community members
- Remove members
- Add and modify pages globally
- Modify properties such as Look & Feel, the Community URL, Community access settings, Community status (active or inactive), and Community expiration

Figure 1-1 illustrates Community creation and management by its members. WebLogic Server users, which can include users from external user stores, can be portal administrators and Community members. Portal administrators create and manage templates and desktops with the Administration Console, and Community members create and manage Communities with the WebLogic Portal Community Tools.

Figure 1-1 Community Creation and Management



End-users can also customize their own views of Communities with the WebLogic Portal Visitor Tools, accessed through links in a Community.

Adding Community Security

In addition to WebLogic Portal's entitlement and delegated administration features, Communities provide an extensible security framework. This security framework, which lets you create security roles called capabilities, gives Community developers control over Community feature access.

Developing Community Features

In addition to developing any type of portal application functionality for Communities, developers can also create Community-specific functionality with the Community framework API.

The following examples are actions you can perform with the Community framework API:

- Access Community and membership information
- Create, activate, and deactivate Communities
- Manage Community membership
- Manage Community security capabilities
- Enable or disable personal pages
- Configure and manage invitations for users to join a Community
- Manage notifications to be sent to Community members

Using the GroupSpace Template

WebLogic Portal provides a predefined Community template called GroupSpace, designed to be used by a team or project management Community. You can create many different instances of GroupSpace. For example, you can create a unique instance of GroupSpace for each team or project. For more information, see the *GroupSpace Guide*.

Communities in the Portal Life Cycle

The portal life cycle contains four phases:

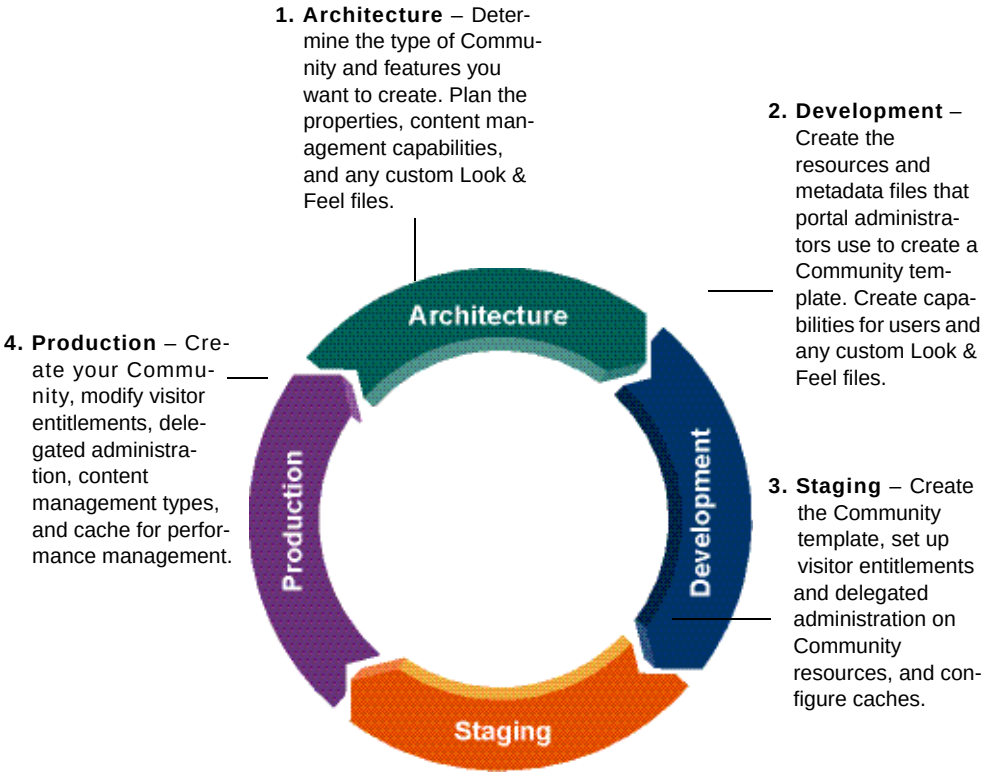
- Architecture
- Development
- Staging
- Production

Use WebLogic Portal to create custom Community applications and templates that suit your Community needs. Like other portal features, creating and managing Communities follows a portal creation and management life cycle. Each phase of the life cycle includes specific tasks and tools.

The tasks in this guide are organized according to the portal life cycle, which implies best practices and sequences for creating and updating Community functionality. For more information about the portal life cycle, see the *Portal Overview*.

Figure 1-2 shows which types of Community-related tasks occur at each phase.

Figure 1-2 Community Tasks in the Portal Life Cycle



Architecture

In the Architecture phase of creating Communities, you determine the type of Community you want to create and plan the resources and functionality you want to be available to that Community. You also determine the properties you want for the Community, such as expiration date and if personal pages are supported. In this phase, if you are creating Community content management capabilities, you also plan the attributes you want to set on Community content. Determine if you want a custom Look & Feel for a Community.

Tools: Use WorkSpace Studio to create Look & Feel files. In the Development phase, you modify or create Look & Feel files, and you can create graphics, CSS files, and JavaScript files with your favorite editors.

The following chapters provide guidance on Community architecture:

- Determining Community Needs
- Community Architecture
- Planning and Designing Communities

Development

Use the Development phase to build the Community features and functionality you designed in the Architecture phase. In Development, you do not create a Community—you create the resources that portal administrators use to create a Community template. The portal administrators and end-users then use the template to create a Community.

For example, in the Development environment you create page flows and JSPs, and then you surface those in portlets. When you create a portal file, you create books, pages, and other portal resources, and you add portlets to your pages. Portal administrators use the resulting `.community` file to create a Community template, and end-users create a Community from that template.

You can also control technical Community configuration details by creating Community metadata files (`.ctmeta` files). Portal administrators, who can use these files to create a Community template, are insulated from configuring technical Community details. You also develop Community functionality, such as registration pages or customized invitation functionality, in addition to any standard WebLogic Portal functionality (personalization, for example).

In the development phase you also define the Community capabilities you need, such as *leader* or *contributor*. You can control Community access and functionality by setting up conditional logic against capability names. If you are using the GroupSpace template, you can also modify the default GroupSpace Look & Feel, or create a new one.

Tools: Use WorkSpace Studio for JSP, portal, portlet, and Look & Feel development. Use your favorite editor to modify the Look & Feel graphics, CSS files, and JavaScript files.

The following chapter provides guidance on Community Development tasks:

- Developing Custom Communities

Staging

In the Staging phase, you use the portal resources created in the Development phase to create the Community templates that portal administrators and end-users use to create Communities. If you developed custom content management functionality for your Community (like that found in GroupSpace), set up the required content properties in staging.

In this phase you also set up visitor entitlements and delegated administration on Community resources and configure cache for performance management. After your application is configured and tested, deploy it to the production environment.

Tools: Use the Administration Console to create a Community template, set up Community content types, configure cache, set visitor entitlements and delegated administration on Community resources, and perform any other configuration tasks. Use the WebLogic Portal production operations utilities and the WebLogic Server Administration Console to deploy your application to the Production environment.

The following chapter provides guidance on Community Staging tasks:

- Setting up Community Infrastructure

Production

In the live Production environment, portal administrators and end-users create Communities based on the Community templates created in the Staging phase. Community members with administrative rights manage the Communities. Portal administrators can also modify visitor entitlements, delegated administration, content management types, configure cache for performance management, and perform other tasks.

Tools: End-users use the WebLogic Portal Community Tools, accessed from a menu inside a Community, to create and manage Communities. End-users also use the WebLogic Portal Visitor Tools, also accessed from a menu, to customize their views of Communities.

Portal administrators can also use the WebLogic Portal propagation tools to propagate Community templates and other portal data back to the Staging environment for testing against Production conditions.

The following chapter provides guidance on Community Production tasks:

- Creating and Managing Communities

Getting Started

This section includes the following topics:

- Prerequisites
- Related Guides

Prerequisites

The *WebLogic Portal Tutorial* walks you through prerequisite tasks, such as creating a WebLogic Portal EAR project, a web application, and a portal. The Javadoc also provides detailed technical information about Communities.

Related Guides

And understanding of how to create portals and portlets is essential for Community development, as is understanding the development and administration tools. Familiarize yourself with the following documents before attempting to build Communities:

- *GroupSpace Guide*
- *Portal Development Guide*
- *Portlet Development Guide*
- *Content Management Guide*

For information on other features, such as personalization and production operations, see the WebLogic Portal e-docs home page.

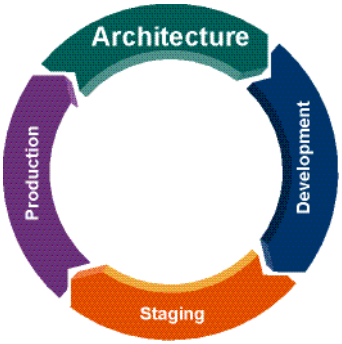
Part I Architecture

Part I includes the following chapters:

- Determining Community Needs
- Community Architecture
- Planning and Designing Communities

In the Architecture phase of creating Communities, you determine the types and characteristics of the Community you want to create. You also plan the resources and functionality you want to be available to the Community.

To learn how the tasks in this section relate to the overall portal life cycle, see the *Portal Overview*.



CHAPTER 2

Determining Community Needs

This chapter helps you determine when to create a Community and when to create regular portal desktops. If you decide to create a Community, this chapter will assist you in deciding if you should create a GroupSpace Community or a custom Community.

This chapter includes the following sections:

Do You Need Communities?

Use the following table to determine if you need to develop a Community or if you should develop a portal desktop.

If you need…	Communities provide…
Tools to have portal features behave differently, depending on Community role For example, you have internal administrators and partner administrators for partner Communities. You need some additional functionality for internal administrators that partner administrators will not see.	**Granular Community context** The Community framework provides Community context information so that you can determine if a user is a member of a certain Community and the role that the user has in the Community. Given these combinations, you can have different functionality available in the same portlet. With regular portal desktops, you can control user access to resources as well. For example, you can use entitlements to control which books, pages, and portlets users see; and you can use the security API and JSP tags to control access to inline content based on security roles. However, assigning users to roles either involves heavy manual, centralized, and always-available portal management, or the development of role-assigning functionality that can be difficult to create and maintain. Communities are advantageous in this case, because Community administrators can assign Community roles to their own members, and the Community gives developers a richer set of information for developing programmatic access to resources.

Custom portal member roles For example, your partner Communities need a number of custom member roles that are constrained to certain application functionality. For example, you need a role for your dealers that lets them administrator information but only view other information.	**Custom Community roles** Using the Community framework, you can define custom roles. These roles are defined in XML and are available within the Community management tools to target capabilities to roles. In regular portal desktops, you only grant or deny access to resources. With Communities, you can control the exact type of access end-users have. For example, some Community users are able to create, edit, and delete content, while other users are allowed to only view content.
To enable others to create custom Communities For example, you built a custom Community for your distributors and now the distributors want to use very similar capabilities in Communities for their resellers. However, there are potentially thousands of Communities for these dealers.	**Community templates** The Community framework includes the ability to create a Community template that includes whatever resources (portlets) you want to include, and roles you define. This template can be used by non-technical end-users to create a new Community. Regular portal desktops provide no capabilities for end-users to create and manage their own desktops.

If you do not need any of those Community-specific features, create regular portal desktops. See the *Portal Development Guide*.

Determining Which Type of Community to Create

If you determined in the previous section that you want to create Communities, you must decide which type of Community to create: a GroupSpace Community or a custom Community.

GroupSpace is a predefined Community application included with WebLogic Portal. GroupSpace contains functionality and tools designed to support a team or project management Community. GroupSpace is built on the Community framework and is an excellent example of a custom Community targeted at the business activity of a team or project collaboration. GroupSpace's features are described in more detail in the *GroupSpace Guide*.

Custom Communities require a full application development effort.

Creating a GroupSpace Community

You can create a fully functional GroupSpace Community in minutes, and basic modifications to the default GroupSpace template are not difficult. GroupSpace includes the following portlets: Mail, Calendar, Contacts, Tasks, Discussion Forums, GS Announcements, GS Issues, GS GroupNotes, GS Links, GS RSS Reader, GS Document Library, CM Browser, GS Enterprise Search, GS Search.

Note: The Mail, Calendar, Contacts, Tasks, and Discussion Forums portlets are also available for use in custom Communities. However, all portlets that begin with GS work only in GroupSpace.

GroupSpace also includes its own registration and invitation functionality, and it triggers a predefined set of events as users interact with it. GroupSpace counts the number of times each of these events occurs.

What's Allowed

You can make basic modifications to GroupSpace, such as removing or rearranging portlets and pages and changing the default e-mail that is sent to invite users to join the Community. You can also add your own portlets to GroupSpace, add remote books, pages, and portlets using Web Services for Remote Portlets (WSRP), and use core WebLogic Portal features such as interaction management and interportlet communication in your custom portlets.

Restrictions

The source code for GroupSpace's page flows is not public, so you cannot modify any of GroupSpace's existing application functionality. For example, you cannot insert events into or modify the registration flow, and you cannot substitute your own notification mechanism for alerting users when events occur or when announcements are made.

Creating a Custom Community

In developing a custom Community, you must develop Community functionality from the ground up: invitation and notification mechanisms, registration, content management configuration, error handling, and application functionality (which includes controlling application behavior based on the security capabilities you define; for example, letting Contributors create content and Viewers only view content).

What's Provided

WebLogic Portal provides a set of Community management tools, which are included in the visitor tools, that let Community administrators create Communities, modify Community properties, configure security rights, and assign security rights to Community members, and let Community members view the Communities to which they belong.

WebLogic Portal also provides a predefined set of collaboration portlets for calendar, mail, tasks, discussion forums, and storing contacts. Inside a Community, the Calendar, Address Book, and Tasks portlets automatically provide Personal and Community tabs for creating items that only that user can see and items that the entire Community can see.

Where to Go Next

- If you decided to create a custom Community, continue with this guide.
- If you decided to create a GroupSpace Community, see the *GroupSpace Guide*.
- If you decided to create a regular portal desktop, see the *Portal Development Guide*.

CHAPTER 3

Community Architecture

This chapter describes the concepts and features that make up the Community framework and describes the logic the portal framework uses to render a Community.

This chapter includes the following sections:

Community Concepts and Features

This section describes the concepts and features in the Community framework. See also "Using the Community Framework API" on page 5-12.

Community Templates

A Community template is the pre-packaged set of resources from which a Community can be built. A Community template resides in the database and allows business users to create Community applications without any coding.

The .community File

A .community file is identical to a .portal file in the development environment, but with a different file extension. A .community is a collection of books, pages, portlets, Look & Feel resources, and other components that eventually become a Community. Portal administrators can create Communities directly with a .community file rather than using a Community template.

Another difference between a .community and a .portal file is that the PortalServlet knows how to render a .portal file as a file-based portal. All Communities, however, must be rendered from the database rather than the file system, so the .community file cannot be rendered as a file. For this reason, the PortalServlet does not know how to render a .community file.

The .ctmeta File

The .ctmeta file is an XML file that lets Community developers preconfigure Community properties, such as the .community file that should be used to create the Community, the registration and error page that should be used, and the Community expiration date. The .ctmeta file makes it easy for portal administrators to create portal templates without having to know technical details about the Community configuration.

Community Membership

When a user joins a Community (becomes a member), information is stored in the database that associates the user with the Community. In addition, the member is assigned to a role, or capability, within the Community (such as owner, contributor, or viewer), which can be assigned to the member when the member is invited to join the Community.

Once part of a Community, members have a CommunityUserContext that lets developers set and get membership information about that user within the Community.

Registration

Registration is the process by which users join Communities and become members. Registration functionality, represented in a landing web page or portal, can include such operations as authentication, letting users create their Community usernames and passwords, and displaying the Communities the members can access.

When you run the WebLogic platform installer, it creates a domain in the `WEBLOGIC_HOME\samples\domains\portal` directory and creates a sample GroupSpace application called GS Example Community. You can see an example of a Registration Page in the Example GroupSpace Community at `http://localhost:7041/groupspace/groupspace.jsp`. Insert your machine name for *<localhost>*.

Communities can be publicly accessible. They do not have to require registration, but a landing page is required when registration is not required.

Community Tools

The WebLogic Portal Visitor Tools, with which users can customize their views of portal desktops, also contain a set of Community tools. The Community tools provide Community management and creation functionality for use in portal applications.

Invitations

Invitations are the means by which you tell users that they have been invited to join Communities. The Community framework provides the means to send invitations through e-mail using JavaMail, with the ability to extend invitations using formats such as instant messaging.

Notifications

Notifications are the means by which users communicate with each other in a Community or are alerted to events that occur within a Community. The notification API lets you develop notification functionality in your Communities.

Notifications, which have a limited life span, may come from other users, web applications, or even the framework itself. When notifications are sent, the Community framework persists them in the database until they are consumed or they expire.

Community Context

Because Communities are designed to serve groups of users with common interests and goals, each Community is scoped to that set of users and has unique characteristics. Community context, represented in the Community framework API, is the collection of Community information, including membership information. Community developers can use Community context to create Community-specific features and functionality. For more information, see the Javadoc for CommunityContext.

Community User Context

Users (Community members) also have context within a Community, which can also be accessed and used by Community developers. The CommunityUserContext API lets you get and set member information and perform many types of Community management tasks.

Security: Capabilities vs. Roles

In Community development, you have the option of using capabilities or roles (or both) to control access to Community resources and functionality.

Capabilities

Communities provide a feature called capabilities. For example, the default capabilities for a Community are “creator,” “owner,” “contributor,” and “member.” A capability is a text label applied to a Community member and stored in the member’s record in the database. (A member can have a different capability in each Community.) The only default security functionality with capabilities is that a Community “creator” can delete a Community, and a Community “owner” can manage a Community. Aside from those two behaviors, capabilities are merely labels. To control access to Community resources and functionality with capabilities, you must get the capability for a member from the CommunityUserContext or ICommunityMemberManager, then set up your security logic on the return value.

For example, if you want to show a button on a page to those only in the Community “owner” role, you would get the current member’s capability from the CommunityUserContext or ICommunityMemberManager, followed by the conditional code that shows the button for “owner” and hides it for everyone else.

Roles

Roles are widely used in WebLogic Portal. They provide a mechanism for controlling user access to administration tools and portal resources. WebLogic Server lets you create global security roles, and WebLogic Portal lets you create visitor entitlement and delegated administration roles. For example, you can create a portal desktop and designate that it can be viewed only by users in the global “admin” role or the “manager” visitor entitlement role.

In addition, roles are open to auditing, and in your code you can simply use an `isUserInRole` method or JSP tag to restrict user access to content or actions.

For Communities, the Administration Console provides a bootstrapping mechanism that creates a default set of Community-specific roles. The default Community roles map to the Community capabilities defined in your application’s `META-INF/communities-config.xml` file. If you choose to secure your Community resources with these roles, you can simply use the `isUserInRole` method/JSP tag to secure Community resources.

If you do not want to use Community roles to secure Community resources, or if you want to use another mechanism as well, you can use capabilities.

For Community development, you must decide whether you want to use capabilities, roles, or both to control access to Community resources and functionality.

How a Community is Rendered

The following describes how the portal framework renders a Community. Error code constants are defined in com.bea.netuix.servlets.manager.communities.CommunityRedirectionHelper.

```
Is the Community public?

    Yes: Is the Community active?

         Yes: Render the Community.

         No: Redirect to error page with error
              REDIRECTION_TYPE_COMMUNITY_DEACTIVATED

    No: Is a user logged in and do they have a membership
        in the Community?

        Yes: Is their Community member record active?

             Yes: Is their Community membership record active?

                  Yes: Is the Community active?

                       Yes: Render Community.

                       No: Redirect to error page with
                          error REDIRECTION_TYPE_COMMUNITY_DEACTIVATED

                   No: Redirect to error page with
                       error REDIRECTION_TYPE_MEMBERSHIP_DEACTIVATED

              No: Redirect to error page with error
                   REDIRECTION_TYPE_MEMBERSHIP_DEACTIVATED

        No: Is public registration enabled?

            Yes: Is a user logged in and do they have a Community member record?

              Yes: Is the member record active?

                   Yes: Redirect to registration page.

                   No: Redirect to error page with
                       error REDIRECTION_TYPE_MEMBER_DEACTIVATED

              No: Redirect to registration page.

            No: Is a user logged in and do they have an invitation
                to this Community?

                Yes: Redirect to registration page.
```

```
No: Redirect to error page with error
  REDIRECTION_TYPE_NO_PUBLIC_ACCESS
```

CHAPTER 4

Planning and Designing Communities

In the Architecture phase of creating a Community, you determine the type of Community you want to create and plan the resources and functionality you want to be available to the Community. You also determine the properties you want for each Community.

This chapter provides guidance on the factors to consider and activities to perform prior to beginning Community development.

This chapter includes the following sections:

Portal Web Project Facets

If you create a portal web project that is designed for custom Communities, do not include the GroupSpace facets in that web project. GroupSpace Communities need to run in their own portal web projects and cannot be mixed with custom Communities. For more information, see the *GroupSpace Guide*.

Planning Community Characteristics

Prior to Community development, make decisions about the following Community characteristics. These characteristics, which are set as properties on each Community, can help you determine your Community design:

- If anyone can access the Community without registering as a member.
- If anyone can register as a member in the Community without first receiving an invitation.
- What will serve as the registration or landing page for a Community, and how it should function (especially in light of the decisions you make on the previous properties).
- If you want to provide the visitor tools for Community members to customize their own views of a Community.
- What will serve as the error page that is displayed when a user is not allowed access a Community; for example, if the user is not a member or the Community is disabled. Also determine how you want that page to behave. For more information, see "Developing a Multi-Purpose Community Error Page" on page 5-8.
- If and when the Community will automatically become inactive or expire.
- If you want to track the number of times members visit the Community is stored in the virtual content repository.
- What, if anything, you want to happen automatically when Communities are created, activated, deactivated, or deleted. For example, if you want a Community content repository to be created when a Community is created. You build this type of functionality in a callback class. For more information, see "Developing Callback Classes" on page 5-10.

Determining Security Needs

Decide which roles, or capabilities, you want to provide in your Communities. By default, the Community framework provides "creator" and "owner" capabilities. You can create any others you need, such as "contributor," "leader," and so on. Also define the types of operations you want to provide for your capabilities, such as CREATE, VIEW, UPDATE, and DELETE.

Creating Look & Feel Resources

If you want to make Community interface resources, such as Look & Feel files, shells that include the visitor and Community tools, and layouts available to developers before they begin development, create those resources first.

For more information, see the User Interface Development with Look & Feel Features chapter of the *Portal Development Guide*.

Determining Content Needs

Plan your content needs in advance of Community development.

If you are going to use content management in your Community, whether almost every object is a piece of content or whether you just want document storage, you may find it convenient and secure to create a repository separate from the standard BEA Repository.

Using a separate repository requires a unique data source and tablespace to keep your Community content separate from regular portal desktop content stored in the BEA Repository (or a BEA-compatible third-party repository). First define your data source by extending your domain with the Configuration Wizard and adding the data source and tablespace to your domain. Later in the staging environment you can configure the new repository to use the new data source.

You can also create a repository and directory structure using a callback class on your Community.

See the *Content Management Guide* for information on working with content.

Determining Custom Community Property Needs

You can develop Community functionality based on custom Community properties that you expect to be present in Communities. You create custom Community properties—properties that provide unique information about a Community—in the WebLogic Portal Administration Console for a selected Community.

Determine which, if any, custom Community properties you want to create, especially for properties beyond what standard Community properties you can access with the Community framework API (for example, `com.bea.netuix.application.communities*` and `com.bea.netuix.servlets.manager.communities*`.

Use the `com.bea.netuix.application.communities.CommunityProperty` class in conjunction with the `CommunityPropertyValue` class to get custom Community properties.

See “Creating Custom Community Properties” on page 7-5 for information on creating these properties with the Administration Console.

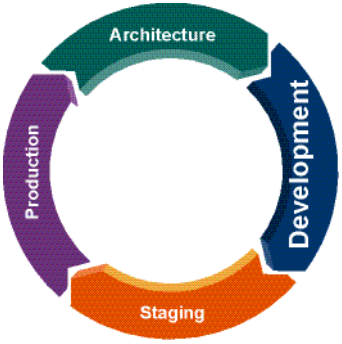
Part II Development

Part II includes the following chapters:

- Developing Custom Communities

The development phase is where you build the Community features and functionality you designed in the architecture phase. In development, you do not create Communities themselves. You create the resources and Community-specific functionality out of which portal administrators create Community templates, which portal administrators and end-users then use to create Communities.

For a view of how the tasks in this section relate to the overall portal life cycle, refer to the *WebLogic Portal Overview*.



CHAPTER 5

Developing Custom Communities

The Development phase lets you build the features and functionality for the Community you designed in the Architecture phase. In Development, you do not create Communities themselves. You create the resources that portal administrators use to create a Community template, which portal administrators and end-users use to create the Community. For example, in the Development environment you create page flows and JSPs, and surface those in portlets. When you create a portal file, you create books, pages, and other portal resources, and you add portlets to your pages. Portal administrators use the resulting `.community` file to create the Community template.

You can create the following Community features and functionality in the Development phase:

- A Community metadata file (`.ctmeta`), which controls technical configuration details, such as registration pages, customized invitation functionality, and other standard WebLogic Portal functionality (for example, personalization).
- Community capabilities, such as *leader* or *contributor.* You can control Community access and functionality by setting up conditional logic against capability names.

Many of the concepts and features involved in developing custom Communities involves portal development features described in the *Portal Development Guide*.

This chapter includes the following sections:

- Developing a Multi-Purpose Community Error Page
- Adding BEA's Collaboration Portlets to Custom Communities
- Adding Community and Visitor Tools to a Community
- Community Security
- Developing Callback Classes
- Extending How Community Invitations Are Sent
- Developing Community Notification
- Using the Community Framework API
- Using Tag Libraries in Your Community
- Adding Interaction Management to Communities

Creating a .community File

A .community file is a .portal file with a different extension. To create a .community file, create a .portal file, then convert it to a .community file. Community files are designed to be used to create Communities. Whether referenced in a Community template metadata (.ctmeta) file for portal administrators to use in creating Community templates, or used directly by portal administrators to create Communities, the .community file contains the books, pages, portlets, and other portal resources that make up the Community.

Another difference between .portal and .community files has to do with the PortalServlet. The PortalServlet knows how to render .portal files as resources from the file system. The PortalServlet, however, does not know how to render .community files from the file system, because Communities must be rendered from the database. When a portal administrator or end user creates a Community, the components from the .Community file are stored in the database for rendering.

When you create a .community file, select a shell that contains the Visitor Tools. These tools provide Community management capabilities.

The Javadoc also provides detailed technical information about Communities.

Tip: For more information on using Community APIs to create a custom Community, see the sample on dev2dev.

Converting Between .portal and .community Files

You can convert .portal files to `.community` files and `.community` files to .portal files. The main intent of this feature is to let you turn existing `.portal` files into resources that portal administrators can turn into Communities. When you turn a portal into a Community, take advantage of the Community framework features that let you build Community functionality.

To convert a .portal file to a `.community` file:

1. In WorkSpace Studio, right-click the **.portal** file in the Navigator view and choose **Convert to Community File**.
2. Click **OK** at the prompt. The file extension changes to `.community`, and portal administrators can use the file to create Communities.

You can convert `.community` files to .portal files using the same procedure.

Creating a Community Template Meta File (.ctmeta)

As a developer, you can make Community template creation easier for portal administrators by creating XML Community template meta files (`.ctmeta`).

Communities are made up of many properties, many of them technical (such as identifying the callback class that is invoked when Communities are deleted). By preconfiguring Community properties in a .ctmeta file, portal administrators have to make fewer decisions and will make fewer mistakes when creating Community templates. The settings you make in the .ctmeta file also ensure a Community behaves the way developers intended it to behave. For example, if a Community is designed for private registration only, you can configure that in the `.ctmeta` file, making that property read-only when portal administrators create a template based on that .ctmeta file.

To see a sample of an existing `.ctmeta` file, create a temporary portal web project that has GroupSpace enabled, and copy the existing `groupspace.ctmeta` file from the shared library (go to the Merged Project view and locate the `<portalWebProject>/groupspace.ctmeta` file) to your project. The `groupspace.ctmeta` file now resides in the `<portalWebProject>/WebContent/` directory in Navigator view.

After you create the new file, modify the Community properties to suit your needs, based on what you have developed.

Following are descriptions of the properties you can set in the .ctmeta file. Within each element you specify the following:

- **The <title> property** – The name that is used for the Community template. The value is a String.
- **The <description> property** – The description of the Community template. The value is a String.
- **The <community-dot-file> property** – The .community file that will become the Community. The value is the relative path to the .community file. For a .community file in the same directory as the .ctmeta file, start the path with a forward slash, such as /mycustom.community.
- **The <callback-class> property** – The value is a fully qualified class name to your custom callback class that can perform operations when Communities are created, activated, deactivated, or deleted.
- **The <expiration> property** – The date that the Community should automatically become inactive. The value is a <dateTime> setting.
- **The <active> property** – Determines if the Community should be activated (available for access) when the Community is created. Set the value to true or false.
- **The <public> property** – Determines if anyone can access the Community without registering as a member. Set the value to true or false.
- **The <public-registration> property** – Determines if anyone can register as a member in the Community without first receiving an invitation. Set the value to true or false.
- **The <personal-pages> property** – Determines if the visitor tools are enabled in the Community to allow members to make customizations to their views of the Community. Community administrators, however, can still access the Community Tools. Set the value to true or false.
- **The <registration-uri> property** – The value is the path to the Community registration page, usually the home page for a Community.
- **The <error-uri> property** – The value is the path to the error page that is displayed when a user is not allowed access a Community; for example, if the user is not a member or the Community is disabled.
- **The <access-tracking> property** – Determines if the number of times members visit the Community is stored in the virtual content repository. Set the value to true or false.
- **The <inviter-invoker-class> property** – The value is a fully qualified class name to your custom inviter class.

The properties in .ctmeta use the following child elements:

- **The <locked>false</locked> element** – If set to false, portal administrators can override the value you provide when creating a Community template with the .ctmeta file. If set to true, portal administrators cannot modify the property value, which is useful for more technical properties such as callback-class.
- **The <value></value> element** – Lets you specify the appropriate property value. The expiration property uses <dateTime> instead of <value>, using the following format: <dateTime>2008-03-25T22:30:00</dateTime>

You can get the values for these Community properties using `com.bea.netuix.application.definition.CommunityDefinition`.

Listing 5-1 contains a sample `groupspace.ctmeta` file.

Listing 5-1 A groupspace.ctmeta File

```
<?xml version="1.0" encoding="UTF-8"?>
<communities-meta
xmlns="http://www.bea.com/servers/portal/communities/meta/1.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/portal/weblogic-portal/8.0
netuix-communities-meta-1_0_0.xsd">
     <title>
          <locked>false</locked>
          <value>Default GroupSpace Community Template</value>
     </title>
     <description>
          <locked>false</locked>
          <value>This is the GroupSpace Community Template.</value>
     </description>
     <community-dot-file>/groupspace.community</community-dot-file>
     <callback-class>
          <locked>true</locked>
          <value>com.bea.apps.groupspace.security.GroupSpaceCallbackImpl
               </value>
     </callback-class>
     <!--
     <expiration>
```

```
            <locked>false</locked>
            <dateTime>2008-03-25T22:30:00</dateTime>
        </expiration>
        -->
        <active>
            <locked>false</locked>
            <value>true</value>
        </active>
        <public>
            <locked>true</locked>
            <value>false</value>
        </public>
        <personal-pages>
            <locked>false</locked>
            <value>true</value>
        </personal-pages>
        <public-registration>
            <locked>false</locked>
            <value>true</value>
        </public-registration>
        <registration-uri>
            <locked>true</locked>
            <value>/portlets/access/registration.portal</value>
        </registration-uri>
        <error-uri>
            <locked>true</locked>
            <value>/groupspace.jsp</value>
        </error-uri>
        <access-tracking>
            <locked>false</locked>
            <value>false</value>
        </access-tracking>
        <inviter-invoker-class>
            <locked>true</locked>
            <value>com.bea.apps.groupspace.invitations.
                GroupSpaceInviterInvoker</value>
        </inviter-invoker-class>
    </communities-meta>
```

Developing Community Registration

When users register and become a member of a Community, they are added to the application's user store, and more extensive functionality can be available to them. For example, Community members can be assigned security capabilities, letting them access resources that would not be available to them if they were unregistered or anonymous users.

Registration is not required for Communities, however. You configure a Community to be public, so that non-members can access the Community.

One example of the interplay between members and non-members is a discussion forum. Non-members can view the discussion threads, but they have to become members if they want to post to the discussions.

A registration URI is required for all Communities, whether or not you want to require registration. A registration can be anything from a JSP to a portal. It points to the starting point for anyone accessing the Community. An example Community with optional registration is a registration portal that contains a registration portlet.

Consider creating a page flow for your registration page. With a page flow, you can control a user's path through the registration process based on such factors as whether or not the user is logged in.

Following is some functionality you might want to build into your registration page:

- Provide username and password creation that adds the user to the user store and makes the user a Community member.
- Provide authentication.
- Let users modify user profile values that are specific to that Community.
- Show users a list of publicly accessible Communities, as well as a list of Communities they are members of or have been invited to. For instructions on how to access multiple Communities to which you belong, see "Accessing Multiple GroupSpace Communities" on page 7-7.

You can use the MemberPicker portlet that ships with WebLogic Portal in your non-GroupSpace Community. You can use the User Picker portlet outside of a Community, in a portal desktop.

For more registration page ideas, create a GroupSpace Community and work through the registration process. See the *GroupSpace Guide* for more information.

Developing a Multi-Purpose Community Error Page

When users try to access a Community but are denied (perhaps because they are no longer members of a Community or the Community is inactive), they see an error page that you provide. An error page can be any resource that can be rendered in a browser, such as a JSP, page flow, or a portal, and is required for all Communities.

Rather than just stating the a Community is inaccessible, error pages can provide useful information, such as links to public Communities. Using the Community framework API (`com.bea.netuix.servlets.manager.communities.CommunityRedirectionHelper`), you can develop useful, multi-purpose error pages.

The most convenient way to use an error page in a portal you develop is to reference it in a `.ctmeta` file. Portal administrators can then use that file to create Community templates without having to provide the error URI themselves.

Adding BEA's Collaboration Portlets to Custom Communities

WebLogic Portal provides the following collaboration portlets that you can use in any Community or portal desktop:

- Calendar
- Mail (supports IMAP and POP)
- Address Book
- Tasks
- Discussion

When used in a Community, Calendar, Address Book, or Task, these portlets provide Community and Personal tabs for storing both Community-level items and personal items in the Community_Repository.

To add the collaboration portlets to your portal EAR and web projects:

1. Add the Collaboration Portlets Application Library facet to your portal EAR project.
   a. In the Navigator view, right-click your portal EAR project and choose **Properties**.
   b. In the Properties view, select **Project Facets**, and click **Add/Remove Project Facets**.

c. In the Add/Remove window, expand **WebLogic Portal Collaboration** and select **Collaboration Portlets Application Libraries**.

d. Click **Finish** and then **OK**.

2. Add the Collaboration Portlets facet to your portal web project.

   a. Perform the same sub-steps above, selecting the **Collaboration Portlets** facet.

After you add the Collaboration Portlets, they are available to add to any portals you create in WorkSpace Studio. They are also available in the portal Library in the WebLogic Portal Administration Console.

Adding Community and Visitor Tools to a Community

WebLogic Portal provides a set of Community and Visitor Tools for use in your Communities and portal desktops.

The Visitor Tools provide a user interface for users to customize their portal desktop environments. The Community Tools, which are built into the Visitor Tools, provide Community management tools that are available to users with administrative rights.

To add the visitor and Community tools to a Community:

1. Create a .community file and select the Visitor Tools Desktop Shell.

   Or, if you want to create a custom version of the Visitor Tools Desktop Shell, which determines the look and behavior of your Community's header region, copy the `/framework/markup/shell/visitorTools.shell` file from the shared library to your file system.

2. Make a copy of the `visitorTools.shell` file. In the new `.shell` file, change the title, description, and markupName attributes to make the shell unique.

3. Configure the `.shell` file to meet your needs, keeping the XML block for the `visitorMenu.portlet` file. For example, you can reference JSPs, page flows, or HTML files to include in the header region. Copy other `.shell` files from the shared library to your file system to see examples of how other shells implement the header region.

For more information about shells, see the User Interface Development with Look & Feel Features chapter in the *Portal Development Guide*.

Community Security

In addition to visitor entitlements, which you can apply to Community resources just like you can to portal desktop resources, Communities provide an extended security feature called capabilities.

Capabilities are simply role names, such as creator, owner, contributor, and visitor that you configure in an XML file; (the `/META-INF/communities-config.xml` file in your portal EAR project). These capabilities are Java Strings that the Community framework automatically adds to the database. In addition to the capability names, you also determine the rights each capability has; for example, CREATE, READ, UPDATE, and DELETE.

Capabilities and their rights are completely free form. You determine the names, and you develop your Community functionality using those capabilities in your conditional logic to control user access to Community resources and features. You also determine how users joining the Community are assigned capabilities. For example, you could develop an invitation mechanism, like GroupSpace's invitation tool, that pre-assigns a capability to a user within the invitation.

The following capabilities are provided by default in all Communities and cannot be removed or renamed:

- Creator – This capability lets the creator delete a Community.
- Owner – This capability provides full management access in a Community

See the "Community Architecture" on page 3-1 for a description of the differences between capabilities and visitor entitlement roles.

Guidelines for Working with Capabilities

Deleting capabilities is not a best practice. If you delete a capability from the XML, you must also delete it from the database. After deleting a capability, any users assigned that capability are still identified as having that capability.

Developing Callback Classes

The Community framework lets you perform customized actions in your Community when Communities are created, activated, deactivated and deleted. You develop these actions using a callback class. For example:

- When a Community is created, you might want to create a folder hierarchy in the virtual content repository and create a few default Community members.

- When a Community is deleted you might want to archive its content and send a note to the former members of that Community to inform them that the Community no longer exists.

Developing a callback class can involve using many parts of the WebLogic Portal API, including the Community framework API.

After you create a callback class, the best way to use it is to add it to a .ctmeta file, so that when portal administrators create a Community template out of the `.ctmeta` file, any Communities created with that template automatically use your callback class.

To create a callback class, extend `com.bea.netuix.application.communities.AbstractCommunityCallback`. This class provides methods for actions you can perform on Community activation, creation, deactivation, and deletion. For best practices information see the Javadoc on the CommunityCallback class.

Extending How Community Invitations Are Sent

The Community framework provides an extensible API for formatting and sending invitations to users to join a Community. The framework provides a mechanism for sending invitations through e-mail using JavaMail. You can use the invitations API to provide different ways of sending invitations to users, such as instant messaging.

For example, GroupSpace provides an invitation tool in the Community Tools that lets you send invitations through e-mail or to users in the user store. When invitations are sent to users in the user store, those users see the invitation when they log into a GroupSpace Community.

See the `com.bea.netuix.application.communities.invitations*` packages in Javadoc for more detailed information on working with invitations. In particular, use the `com.bea.netuix.application.communities.invitations.spi` package for extending the default invitation functionality to include other mechanisms, such as instant messaging.

Developing Community Notification

Notifications let you alert users when certain events occur. For example, you can notify users when content is added to the virtual content repository or when the status on a task changes. Notifications are directed to one or more users in your application's user store, and can be handled in a variety of ways.

For example, notifications are used for announcements in GroupSpace. A user creates an announcement and sets it to be broadcast on a specific date. At the specified date and time, the notification framework sends the announcement, and the users that received the announcement see a link to the announcement when they log into GroupSpace.

Notifications are not a Community-specific feature. The notification framework provides a communication mechanism you can implement in any portal. In particular, you can configure the portal framework to send messages between portlets when notifications are triggered.

See the following packages in Javadoc for detailed information on working with notifications:

- The `com.bea.netuix.application.communities.notifications` package
- The `com.bea.netuix.application.notifications` package
- The `com.bea.netuix.application.manager.persistence.NotificationManager` package

Using the Community Framework API

The Community framework API provides a set of classes and methods for developing Community-specific functionality in you custom and GroupSpace Communities.

The programmatic tasks you can perform with the Community framework API are useful only inside a Community—in the "context" of a Community. For example, if you create a portlet that retrieves a list of all the Communities to which the user belongs, that portlet would not function in a regular portal desktop, where there is no Community context.

You can use the Community framework API in any area of a Community application that lets you use Java code, such as in JSPs, page flows, callback classes, and backing files.

See the following packages in Javadoc for detailed information on operations you can perform with the Community framework API:

- The `com.bea.netuix.application.communities` and `com.bea.netuix.servlets.manager.communities` packages – These packages and their subpackages provide the core set of API functionality for developing Community functionality. Functionality you develop with these packages works in the context of any Community.
- The `com.bea.netuix.servlets.manager.communities.CommunityRedirectionHelper` class – A helper class to let you redirect users who try to access Communities that are disabled.
- The `com.bea.netuix.application.manager.ICommunity` class – This class is application scoped, as compared to CommunityContext, which is web application scoped. With ICommunity you can, for example, add a user to multiple Communities.

- The `com.bea.netuix.application.definition.CommunityDefinition` class – This class contains related Communities information associated with a Community desktop. The class is basically an extension of the DesktopDefinition, but it contains extra Community-specific information, such as the CommunityDefinitionId, created date, Community description, desktop path, error URI, callback class name, expiration date, a pointer to a parent community (if it exists), a set of properties associated with the Community, the registration page URI, the name of the webapp that contains the Community, and flags to determine if the Community has access tracking enabled, if the Community is active, if it is global, if it is public, if it is a template, and if public registration is enabled.
- The `com.bea.netuix.application.manager.persistence.*` interfaces – This class contains EJB interfaces, including those that let you perform Community-related operations.
- The `com.bea.netuix.application.communities.messaging` package – Messaging can be used for activities that involve sending messages to users, such as reminders of tasks due, announcements, notifications, and invitations. Use the class to get the messaging address and the messaging address type: EMAIL, INSTANT_MESSAGE, and CUSTOM.

Using Tag Libraries in Your Community

During the Development phase, you can use tag libraries to add features to a GroupSpace Community, a custom Community, or a portal web application. This section discusses some of the many tag libraries that you can use with your communitiy application.

- The `ActiveMenus` JSP Tag Library
- The `DragDrop` JSP Tag Library
- The `DynamicContent` JSP Tag library
- The `UserPicker` JSP Tag library

See the *Portlet Development Guide* for detailed instructions on adding, configuring, and using these tag libraries.

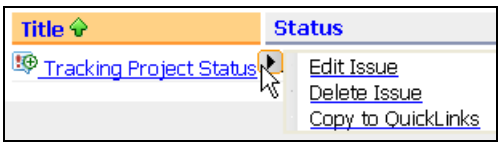
Using the ActiveMenus Tag Library

During the Development phase, you can add a resource called ActiveMenus to a GroupSpace Community, a custom Community, or a portal web application. The ActiveMenus JSP tag library lets you set up a popup menu that displays when the mouse hovers over specific text. An

`activemenus-config.xml` file controls the contents of each menu. The `activemenus_taglib.jar` file contains the ActiveMenus tag library.

By default, a GroupSpace Community has ActiveMenus enabled, so you only need to configure the ActiveMenus tag (see the *Portlet Development Guide*). See Figure 5-1 for an example of the ActiveMenus tag in a GroupSpace Community.

Figure 5-1 ActiveMenus in the GS Issue Portlet



You can tie a user's capability to the ActiveMenu that you see when you hover your mouse over an item (an Issue, for example) and hover over the arrow that appears. In this example, if your assigned capabilities include the ability to delete items, you will see the **Delete** choice, as shown in Figure 5-1.

Tip: You do not need to perform the following steps if you have a GroupSpace Community; ActiveMenus are enabled by default for GroupSpace Communities.

See the *Portlet Development Guide* for instructions on enabling and configuring Active Menus in a custom Community and how to use the following tags with the `ActiveMenus` tag:

- The `typeInclude` Tag
- The `type` Tag
- The `typeDefault` Tag
- The `menuItem` Tag

Using the DragDrop Tag Library

During the Development phase, you can use the `DragDrop` JSP tag library to enable drag and drop functionality in a GroupSpace Community, a custom Community, or a portal web application. You must identify draggable objects that are displayed on a JSP, and identify drop zones that are configured to react to a dropped draggable object. The drop zones react by triggering Page Flow actions, calling JavaScript functions, or posting data to a servlet.

See the *Portlet Development Guide* for instructions on enabling and configuring Drag and Drop in a custom Community and how to use the following tags with the `DragDrop` tag library:

- The `dragDropScript` Tag
- The `draggableResource` Tag
- The `resourceDropZone` Tag

Using the Dynamic Content Tag Library

During the Development phase, you can use the `DynamicContent` tag library to quickly update parts of a JSP page in a GroupSpace Community, a custom Community, or a portal web application.

The `DynamicContent` tags let you use an AJAX request to update part of a JSP page within a Page Flow-based portlet. The tags allow parts of the page to be updated without performing a full portal request. These AJAX requests are smaller and faster than full portal requests, and therefore provide a more responsive user experience when interacting with a portal application.

These tags are incorporated into standard Page Flow-based portlet development and can help create advanced user interface features that improve a user's portal experience.

See the *Portlet Development Guide* for instructions on using dynamic content in a custom Community and how to use the following tags with the `DynamicContent` tag library:

- The `container` Tag
- The `containerActionScript` Tag
- The `executeContainerAction` Tag
- The `parameter` Tags

See dev2dev for sample code and utilities contained in a `sample.zip` file for the Dynamic Content tag library.

Using the UserPicker Tag Library

During the Development phase, you can use the `UserPicker` tag library to add a form button to a JSP page in a GroupSpace Community, a custom Community, or a portal web application.

The `UserPicker:popupButton` tag provides the developer with the ability to add a form button to a JSP page which opens a popup window that displays a list of current users. You can select a

user from this list. The name of the selected user is populated into a specified form field on the parent window.

See the *Portlet Development Guide* for instructions on using the following `UserPicker:popupButton` tag attributes:

- The `inputId` Tag
- The `inputTagId` Tag
- The `buttonImage` Tag
- The `atnProviderName` Tag

Tip: When the `UserPicker:popupButton` tag is used in a Community, the Community members are listed, rather than users.

Adding Interaction Management to Communities

In addition to developing Community functionality in your Communities, you can also use existing WebLogic Portal interaction management features, such as personalization, campaigns, and events. For more information, see the *Interaction Management Guide*.

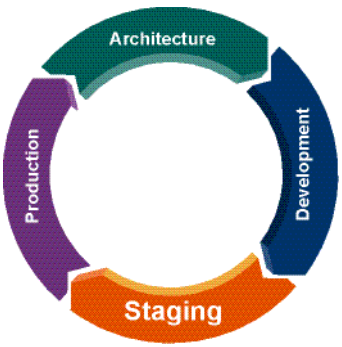
Part III Staging

Part III includes the following chapters:

- Setting up Community Infrastructure

In the staging phase, you use the portal resources created in development to create the Community templates with which portal administrators and end-users create Communities. If you developed custom content management functionality for your Community, set up the required content properties in staging. In this phase you also set up visitor entitlements and delegated administration on Community resources and configure cache for performance management in this phase. After your application is configured and tested, you deploy it to the production environment.

For a view of how the tasks in this section relate to the overall portal life cycle, refer to the *WebLogic Portal Overview*.



CHAPTER 6

Setting up Community Infrastructure

In the staging phase, you use the portal resources created in development to create the Community templates with which portal administrators and end-users create Communities. If you developed custom content management functionality for your Community, set up the required content properties in staging.

In this phase you also set up visitor entitlements and delegated administration on Community resources and configure cache for performance management in this phase. After your application is configured and tested, you deploy it to the production environment.

This chapter includes the following sections:

Creating a Community Template

A Community template lets users create a Community that has specific features and characteristics. As a portal administrator, you create Community templates for end users that have specific characteristics, depending on Community needs. For example, the template for a financial Community would have different books, pages, and portlets than the template for a music Community.

Because of the unique characteristics of each Community, Community templates must be well planned from the design phase through the development phase. Once you enter the production environment, a Community must look and behave in the desired way.

You can create Community templates based on resources provided by your development team (`*.ctmeta` or `*.portal` files) or based on existing books, shells, and Look and Feel files in the portal library. You can also create a new book that your template will use.

Perform the following steps to create a Community template:

1. If necessary, create a new portal in the ProductNameShort.
2. In the Portal Resources tree, expand the **Portals** directory, select your Portal, select **Templates**, and select **Community Templates**.
3. On the Community Templates page, click **Create Community Template**. The Create Community Template wizard appears.
4. In Step 1 of 6 in the wizard, select the source for your template and click **Next**. Use **Search** or **Show All** to display the available resources.
   - **From a Community template metadata file** – Lets you select any `*.ctmeta` file your development team has created. These files contain preconfigured Community properties, making Community creation easier when more technical properties have been preset. This is the recommended method and is the most common method to create a Community template.
   - **Select resources in the Library** – Lets you create a Community using an existing book, or a book you create yourself, in the portal Library. Make sure the book you select was designed to be used in a Community.

     Note: If you create a new book in the template-building process, after the template is created you must go into the portal Library and configure the book with the resources you want it to have.
   - **From a .portal file** – Lets you create a Community using an existing `.portal` file created by your development team. Make sure the `.portal` you select was designed to be used as a Community.
5. In Step 2 of 6, select the appropriate resource and click **Next**.
6. In Step 3 of 6, enter the following information and click **Next**:
   - **Title** (required) - The title that appears when selecting the template.

- **Description** (optional) - The description appears when an end user creates a Community using this template. It helps the Community creator understand which template he is selecting.
- **Default Shell** (required) - Select a shell that includes the Community Tools. Confirm this with your development team.
- **Look and Feel** (required) - Select a look and feel designed for or compatible with the Community.
- **Add this template to the Library** (optional) – Select the check box if you want to use the template to create additional Communities in the current portal web project.

7. In Step 4 of 6, set any available properties for the Community and click **Create Template**. See "Community Properties Reference" on page 6-3 for property descriptions. You can also click **Review Properties** to review what you enter here.
8. In Step 5 of 6, if you have an existing Community template you are prompted to resolve conflicts. Some of the resources your template uses (such as books, pages, and portlets) might have the same internal names as existing resources. The resources might be exactly the same, or new resources might have matching names. This window lets you decide what to do with the resources your template is using; click **Next** after you determine how to resolve the conflict. Review the Community properties and click **Create Template**. (If you clicked **Create Template** in step 7, you can skip this step.)

   Note: XML markup refers to the basic definition and configuration of the portlet, which does not include end-user customizations.
9. In Step 6 of 6, view the summary of the created template and click **Finish**. Portal administrators and end users can now create a Community based on that Community template.

Community Properties Reference

Community properties can be set in the following ways:

- In development, when developers create `.ctmeta` files that portal administrators use to create Community templates and Communities.
- In production, when portal administrators create Community templates based on library resources or .portal files.
- In production, when end users create Communities based on existing Community templates.

The following are descriptions of Community properties.

- **Title** – The default name of the Community that appears in the user interface. The title can be overridden by the localized Community title.
- **Description** – The default description of the Community that appears in the user interface. The description can be overridden by the localized Community description.
- **Allow non-members to access this community** – If set to true (checked), anyone can access the community without registering as a member.
- **Allow anyone to register for community membership** – If set to true (checked), anyone can register as a member in the Community without first receiving an invitation.
- **Allow people to customize their view and add personal pages** – If set to false (unchecked), the Visitor Tools are disabled in the Community. Community administrators, however, can still access the Community Tools.
- **Registration Uri/Page** – The path to the Community registration page, usually the home page for a Community.
- **Error Uri/Page** – The path to the error page that is displayed when a user is not allowed access a Community; for example, if the user is not a member or the Community is disabled.
- **Expiration Date** - The date when the Community will automatically become inactive.
- **Make this community active now / Status** – If set to Inactive, only Community members with an administrative capability can access the Community.
- **Track member visits to this community** – If set to yes, the number of times members visit the Community is stored in the virtual content repository.
- **Callback Class** – The name of the class (including the package path information) that performs special tasks when the Community is created, activated, deactivated, or deleted. For example, the GroupSpace callback class is `com.bea.apps.groupspace.security.GroupSpaceCallbackImpl`. Your development team might provide its own callback class.
- **Enable Tree Optimization** – If set to true, improves Community performance. For more information on this feature, see Designing Portals for Optimal Performance in the *Portal Development* Guide.

Deploying Communities

The WebLogic Portal propagation tools do not support the propagation of Communities. The tools support only propagation of Community templates. For this reason, do not create Communities in the staging environment. Create them in the production environment.

To test your Communities prior to making them public in the production environment, create them as inactive. The Community creator can then configure and test the Community before activating it.

For information on deployment and propagation, see the *Production Operations Guide*.

Part IV Production

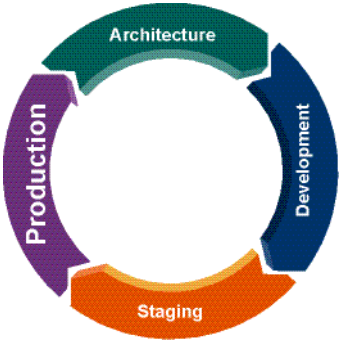
Part IV includes the following chapters:

- Creating and Managing Communities

In the live production environment, portal administrators and end-users create Communities based on the Community templates created in staging, and Community members with administrative rights manage Communities. Portal administrators create new Community templates, create Communities, modify visitor entitlements, delegated administration, content management types, configure cache for performance management, and other relevant tasks.

Portal administrators can also use the WebLogic Portal propagation tools to propagate Community templates and other portal data back to the staging environment for testing against production conditions.

For a view of how the tasks in this section relate to the overall portal life cycle, refer to the *Portal Overview*.



CHAPTER 7

Creating and Managing Communities

In the live production environment, portal administrators and end-users create Communities based on the Community templates created in the Staging phase, and Community members with administrative rights manage Communities. Portal administrators can also modify visitor entitlements, delegated administration, content management types, configure cache for performance management, and other relevant tasks. If you are creating a GroupSpace Community, rather than a custom Community, you should configure the ActiveMenus tag in the Production phase. See "Using Tag Libraries in Your Community" on page 5-13 for more information.

Portal administrators can also use the WebLogic Portal propagation tools to propagate Community templates and other portal data back to the staging environment for testing against production conditions.

This chapter includes the following sections:

Creating a Community

As a portal administrator, you can create Communities with the WebLogic Portal Administration Console. In order for end users to create their own Communities, at least one Community must exist that includes the Community Tools. The Community Tools provide the menu that lets administrative Community members create new Communities.

You must create Communities in the production environment rather than in the staging environment, because the WebLogic Portal propagation tools do not propagate Community instances.

When a user creates a Community, that user automatically becomes the Community creator and owner with full management rights in the Community.

Tip: If you make a Community inactive when you create it, you or any Community member with administrative rights can test and configure the Community before you activate it. If your Community functionality depends on the presence of custom Community properties, also define those before activating a Community. (See "Creating Custom Community Properties" on page 7-5.)

This section describes how to create a Community with the ProductNameShort. The steps are similar if you create a Community using the Community tools inside of a Community.

One reason to create a Community with the WebLogic Portal Administration Console is to provide an initial Community that contains the Community tools, where end users can create their own Communities.

1. In the ProductNameShort, choose **Portal > Portal Management**.
2. In the Portal Resources tree, verify that the correct web application is selected. If it is not, click **Update WebApp** and select the web application you want.
3. Create a portal if you did not already create one.

4. In the Portal Resources tree, expand the **Portals** directory, expand the portal you created, and select the **Communities** directory.
5. In the Browse tab, click **Create Community**. The Create Community wizard appears.
6. In Step 1 of 5 in the wizard, select the source for your template and click **Next**.
   - **Use a Community Template** – Lets you select an existing Community template to create the Community.
   - **Select resources in the Library** – Lets you create a Community using an existing book (or a book you create yourself), an existing shell, and an existing look and feel in the portal Library. Make sure the resources you select were designed to be used in a Community. For example, the shell should contain the Community Tools. See the *Portal Guide* for more information on library resources and `.portal` files.
   - **Select a .community or .portal file** – Lets you create a Community using an existing `.community` or `.portal` file created by your development team. If you use a `.portal` file, make sure it was designed to be used as a Community.
7. In Step 2 of 5, click **Search** or **Show all** to display the available choices. Select the appropriate resource and click **Next**.
8. In Step 3 of 5, enter the following and click **Create Community**.
   - **Title** - The title that appears for the Community in the user interface.
   - **Description** - The description that appears when the Community is selected.
   - **Partial URL** - The text that serves as the end of the URL for the Community.
   - Select and enter the remaining properties for the Community, as described in "Community Properties Reference" in the Setting up Community Infrastructure chapter.

   You can also click **Review Properties** to review what you entered.
9. If you clicked **Review Properties** in step 8, review the information and click **Create Community**. You can also enter the following information:
   - Set the date that the Community should expire. If you do not want the Community to expire, leave the field blank.
   - Select the **Make this community active now** option to make the Community available immediately. If you want to configure and test the Community before making it active, deselect this option.
10. In Step 5 of 5, click **Go to community** to view it and then click **Finish**.

To view your new Community, you can also select it in the Portal Resources tree and click **View Community**.

Activating a Community

To activate an inactive Community, which allows users to access the Community:

1. In the Portal Resources tree, select the Community you want to activate by expanding the **Portals** directory, selecting your Portal's directory, selecting the **Communities** directory, and selecting your Community.
2. On the Summary tab, click **Status & Expiration**.
3. In the Status & Expiration dialog, select **Active** from the **Status** drop-down field and click **Update**.

Modifying and Deleting a Community Template

When you modify a template, new Communities created with that template take on the new template properties and characteristics. However, existing Communities that were created with the original template do not change. If you want the changes to apply to existing Communities, you must modify those Communities directly.

1. In the Portal Resources tree, expand the **Portals** directory, your portal's directory, the **Templates** directory, the **Community Templates** directory, and select your template.
2. On the **Details** tab, click any of the links or buttons to modify the template. Refer to the Community Properties Reference section in the Setting up Community Infrastructure chapter for details.
3. To delete a Community template, expand the **Portals** directory, select your Portal's directory, the **Templates** directory, and select the **Community Templates** directory. Select the **Delete** check box next to the template and click **Delete**. Communities that were based on that template are not affected.

Communities should use a shell that contains the Visitor Tools.

Modifying Community Properties

After you have created a Community, you can modify its properties in the ProductNameShort. After your modifications, the Community immediately uses the new settings.

1. In the Portal Resources tree, expand the **Portals** directory, select your Portal's directory, select the **Communities** directory, and select your Community.
2. On the **Summary** tab, click any of the links or buttons to modify a property. See "Community Properties Reference" on page 6-3 and "Managing Community Members" on page 7-8 for more information.

Communities should use a shell that contains the Visitor Tools.

Creating Custom Community Properties

After you create a Community, you can add custom properties to it to uniquely identify the Community and its characteristics. Your developers can then develop programmatic functionality using those properties.

To create a custom Community property:

1. In the Portal Resources tree, expand the **Portals** directory, select your Portal's directory, select the **Communities** directory, and select your Community.
2. Select the **Custom Properties** tab.
3. Click **Add Property**.
4. In the Create Custom Property dialog, enter the property name and value. To add more than one value, click **Add Another Value**.
5. Click **Save**.

You can also modify or delete a custom property by clicking **Edit** or selecting the **Delete** check box and clicking **Delete** next to the property.

Setting Delegated Administration Rights on Community Resources

You can control administrative access to Community resources by assigning delegated administration roles to resources, the same way you do for portal desktop resources.

Setting Visitor Entitlements on Community Resources

You can control end user access to Community resources by assigning visitor entitlement roles to resources, the same way you do for portal desktop resources.

Note: Be careful when setting entitlements on Community resources. Entitlements applied to components (books, pages and portlets) of the Community desktop at the definition level may result in some or all of the components not being visible to Community members.

Creating Localized Community Titles and Descriptions

You can provide localized text for the title and description of your Communities. When you provide localized text, and a browser is set to use a language you provide, the localized title and description of that Community appear in that browser, either in the ProductNameShort or in the Community itself. For example, in GroupSpace, localized titles appear in the GroupSpace header. For more information, see the *GroupSpace Guide*.

If you want to use double-byte characters, such as Chinese, make sure your system is configured to support this.

You can also use this feature to provide an alternative Community name and description in English, changing them as often as you like.

1. In the Portal Resources tree, expand the **Portals** directory, select your Portal's directory, select the Communities directory, and select your Community.
2. Select the **Title & Description** tab and click **Add Localized Title**.
3. In the Add a Localized Title & Description dialog, enter:
   - **Language** (required) – The two-letter code for the language. For example, `zh` for Chinese.
   - **Country** (optional) – The name of the country.
   - **Variant** (optional) – The two-letter language variant in uppercase, such as US for the United States English, or TW for Taiwan Chinese.
   - **Title** (required) – The localized title you want to provide.
   - **Description** (optional) – The localized description you want to provide.
4. Click **Create**.

If you want to modify an existing title and description, such as changing the English text, click the language link and modify the title and description.

Deactivating or Deleting a Community

If you want to prevent users from accessing a Community, either deactivate or delete the Community. Deleting a Community permanently removes it from the database. Deactivating a Community lets administrators access it for configuration, archive, or reactivation in the future.

Deactivating a Community

To deactivate an active Community, which prevents users from accessing the Community:

1. In the Portal Resources tree, select the Community you want to deactivate.
2. On the Summary page, click **Status & Expiration**.
3. In the Status & Expiration dialog, select **Inactive** and click **Update**. Or if you want to deactivate the Community on a specific date, select **Active**, set the **Expiration Date** for when it should automatically be deactivated, and click **Update**.

The callback class assigned to the Community determines what occurs when the Community is deactivated. For example, notifications can be sent to users when a Community is deactivated.

Deleting a Community

Deleting a Community permanently removes it from the database.

Perform the following steps to delete a Community:

1. In the Portal Resources tree, expand the **Portals** directory, select your Portal's directory, and select the **Communities** directory.
2. On the Browse tab, select the **Delete** check box next to the Community name.
3. Click **Delete**.

Accessing Multiple GroupSpace Communities

If you belong to more than one GroupSpace Community, you can log in once and access Communities you have joined, Communities to which you are invited to join, and public Communities that anyone can join.

Perform the following steps to access the GroupSpace Communities to which you belong and view any invitations:

1. At the GroupSpace Home page, enter your username and password to log into your GroupSpace Community.
2. Click a Community to which you belong.
3. At the main Community page, click **Home**.
4. Since you have already logged in, you can click an invitation link to view it or click a Community link to access another Community.

Managing Community Members

You can modify the capability rights of members or remove members from Communities in the ProductNameShort.

Capabilities, such as creator, owner, contributor, and viewer, dictate the level of access members have to Community resources. You can change the capabilities of Community members. For example, to give a user administrative access to a Community, change the user's capability to owner. The creator and owner capabilities are provided automatically in all Communities. Any additional capabilities are provided by your development team.

You can use the MemberPicker portlet that ships with WebLogic Portal in your non-GroupSpace Community. You can use the User Picker portlet outside of a Community, in a portal desktop. A Portal System Administrator has the ability to see all users in a portal desktop.

The following procedures show you how to manage members with the ProductNameShort.

Modifying Member Capabilities

1. In the Portal Resources tree, expand the **Portals** directory, select your Portal's directory, the **Communities** directory, and select your Community.
2. Select the **Members** tab.
3. Click **Edit** next to the user whose capabilities you want to change.
4. In the Membership Details dialog, select the capability you want, and click **Save**.

Removing Members from Communities

Removing a member does not delete the member from the user store—it only removes the member from the Community. On the **Members** tab, select the **Remove** check box next to the member you want to remove, and click **Remove**.

Propagating Communities

You cannot propagate a Community back to the Staging or Development phases. You can only propagate Community templates.

For information on deployment and propagation, see the *Production Operations Guide*.