Site Design Introduction

This introduction provides a high-level look at what Site Design Help offers. Here, you will find an overview of Site Design, links to important Help topics pertaining to Site Design, and copyright information.

NXT 3 uses templates as the basis for serving pages to a browser. NXT 3 templates are HTML pages, XSLT, JavaScript, CSS, and GIF and JPEG images. Some of these contain NXT 3 Replacement Variables and references to Java applets. NXT 3 includes a default set of templates that you can use and customize to meet your needs, or you can design your own pages and templates to use NXT 3 technology.

You should have an understanding of both HTML and JavaScript to effectively study and modify the templates.

Site Design Help aids you in modifying the existing templates, as well as providing you with the necessary information to create your own templates. The following sections help you to learn more about NXT 3 site design.

- The Master Glossary contains many terms that may be new or unfamiliar to you. You'll also find green-colored links throughout Site Design Help. These open up a small window containing the underlined word's definition.
- Tasks provide the steps necessary to complete a given objective. There is also a Critical Tasks document that provides links to tasks that are essential for Site Design.
- Concepts information provides background on key aspects of the task or software. They are meant to help you understand the product better.
- See Reference topics to get more detail about a specific element on a window or screen.
- Examples provides help for common or complex tasks by supplying sample data and information.

Document Information and Copyright Notice

Document Name	Version	Date
Site Design	3.4	May 2002

Copyright Notice

NextPage, Inc.

3125 West Executive Park Way

Lehi, UT 84043

U.S.A.

Site Design Tasks

Each task is a step-by-step explanation for how to complete a process required to fulfill an objective. Tasks are separated into groups of related tasks that may further be categorized by the degree of importance a task has in helping users meet objectives. Critical tasks are tasks deemed essential to effectively using Site Design. Supporting tasks are tasks that may be performed while completing critical tasks. For experienced users, supporting tasks explain already familiar features, so they may rely more exclusively on the critical tasks.

Site Design contains the following tasks:

- Critical Tasks links to essential tasks for doing site design.
- Formatting Tasks have to do with using cascading stylesheets and XSL stylesheets, which control the presentation of HTML and XML documents.
- Search Form Tasks provide instructions on how to create and incorporate search forms into a site.
- Search Results Tasks are things that you can do with documents returned from a search (like highlighting the search words and providing navigation from one match to the next.
- Security Tasks guide you in setting up access control for sites and views within a site.
- Site Tasks deals with getting content into the site, using NXT 3 components, and creating one or more sites without using the existing templates.
- Template Tasks provide information on using the NXT 3 templates, as well as information on how to use or not use frames for a site.
- Table of Contents Tasks involve steps necessary to create or change the Java or HTML table of contents for the site.
- View Tasks give the steps necessary to set up and edit one or more views on a site.

Site Design Critical Tasks

The following are tasks that are essential when doing site design for new objects or for modifying existing ones:

Tasks for Working with New Objects

- Creating a new site.
- Creating new templates.
- Creating a new view for the site.
- Adding content to the site.
- Setting up access control for the site and view.
- Creating a frameset for the site.
- Setting up a table of contents.
- Creating a document frame.
- Creating a search results page.
- Creating a search form.
- Registering a search form.

Tasks for Modifying Existing Objects

- Working with template files.
- Editing the view.
- Changing site security.

Site Design Formatting Tasks

Site_Design_Introduction

These tasks give you instructions and information on using cascading stylesheets and XSL stylesheets with NXT 3. These files control the presentation of HTML and XML documents.

The following tasks help you with document formatting:

- Creating a CSS
- Implementing CSS
- Creating an XSL Stylesheet
- Using XSL Stylesheets

Creating a Cascading Stylesheet

Site Design Introduction

Cascading style sheets (CSS) are used to define how a document is to be formatted. You can use them with both HTML and XML documents.

Internal Stylesheet

You can define the style that a document uses inside each document individually. If you use this method, then when you make a change to the style, you must make the change in every document.

To include formatting instructions directly in the file, declare `<style type="text/css">` directly in the HEAD of the document (or root node for XML documents), followed by the same formatted style instructions as for an external CSS (for instance, `body { color: black; background: white; }`). Then end the section with the end style tag: `</style>`.

Or, you can include formatting directly in an element using the STYLE attribute (for instance, `<H1 STYLE="color: green">Title</H1>`).

External Stylesheet

The external stylesheet is where you can see the real power of stylesheets. In an external stylesheet, each document references an external file that contains the formatting instructions. In this way, you can make a formatting change to a single stylesheet file, and have it affect every document that uses that stylesheet.

If you need to create an external CSS, complete the following steps:

1. Create a text document in your templates directory (or another place where it can be easily linked to by your documents) and name it something appropriate.

   Note: It is customary to use the CSS extension for stylesheets, but you may use whatever extension you want.
2. Type each element that you want to format on a different line. For instance, TITLE, P, TD, H1, etc.
3. Follow the element with curly braces:

   `{ }`
4. Type the CSS formatting information, ending each formatting option (even the last one) with a semicolon (;).

   See W3C's web site for information about CSS formatting.
5. Save the file, and view the results for the site.

See Also

Implementing CSS

Implementing Cascading Style Sheets

Site_Design_Introduction

Cascading style sheets (CSS) are used to define how a document is to be formatted. Both HTML and XML can make use of CSS. CSS formatting can be included within each document, but the real power comes from having a separate CSS that many files can reference.

To implement an external CSS file for content in your NXT 3 site, do the following:

1. Open the HTML or XML files.
2. Add a reference to a CSS file.

   For HTML, add the reference in the HEAD section of the document.

   ```
   <stylesheet type="text/css" href="#!--
   #STYLESHEETS:style.css --#"/ >
   ```

   For XML, include the reference in the root node's template block.

   ```
   <?xml-stylesheet type="text/css" href="#!--
   #STYLESHEETS:style.css --#"?>
   ```

   Note: This code references a CSS file named style.css that is stored in the Templates directory for the site. The stylesheet file must be located in the Templates directory.

If the CSS file has already been created and is located in the templates directory for the site, you should be able to view the newly-styled content.

See Also

Creating_a_CSS

NXT_3_Replacement_Variables in *Requesting Content from NXT 3 Help*

Creating an XSL Stylesheet

Site_Design_Introduction

In order to create an XSL stylesheet, you should have some knowledge of XML and of what stylesheets can do.

Before you can start, you need to have an XML document to apply formatting to. You also need to create a stylesheet file. The name of the file is not important. NXT 3 typically uses the XSL extension to signify that the file is an XML stylesheet.

To create a stylesheet, complete the following steps:

1. Type the XML declaration at the top of the file.
2. On the next lines, add the namespace declaration and then the output information.

   Note: NXT works best with the UTF-8 output encoding.
3. Now, add a template block to match the root node of the XML document:
4. Finally, put in the stylesheet closing tags.

When you are done, the file could look something like the following:

```
<?xml version="1.0"?>
 <xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:output encoding="UTF-8" method="html" indent="no" />

 <xsl:template match="/">
 </xsl:template>

</xsl:stylesheet>
```

You can view the raw XML output by having an empty xsl parameter. For example: `?f=xhitlist&xhitlist_xsl=&xhitlist_q=hello`. See Document_Parameters in *Requesting Content for NXT 3 Help* for more information.

Adding More Formatting

To get a more formatted document, you can add code between the "xsl:template" section (for instance, `<h1><xsl:value-of select="Document/Title" /></h1>`) to more precisely control how the document is formatted.

See Also

Using_XSL_Stylesheets

XSL_Transformations_Reference

Using XSL Stylesheets

Site_Design_Introduction

XSL Stylesheets transform XML to HTML dynamically, when the user requests a page.

You can add any HTML formatting necessary to make the end result appear in the Web browser the way that you want it to.

XSL stylesheets can be used to match patterns in an XML document, and have logic added to them that treats items differently depending on the properties of that item.

Pattern matching is one of the basic building blocks of an XSL stylesheet. To employ pattern matching in your stylesheet, you need to know the tags that are used in the XML documents that you want to format.

To use XSL stylesheets in your documents, do the following:

1. Create a basic XSL stylesheet.
2. Add as many "template match" sections as you need. For instance, the following code matches the "Document" and the "Image" elements of the root node, and outputs the type that each is:

```
<xsl:template match="Document">
This is a document.
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="Image">
This is an image.
<xsl:apply-templates/>
</xsl:template>
```

3. For an XML document to use the stylesheet, you must specify the stylesheet in the root node of the document:

```
<?xml-stylesheet
type="text/xsl"
href="style.xsl"?>
```

4. Put the stylesheet and XML documents online and save the site changes.
5. View the changes in the site.

See Also

Creating_an_XSL_Stylesheet

Site Design Search Form Tasks

Site_Design_Introduction

These tasks provide directions on how to create a search form, get it included in your NXT 3 site, and the various options that you can set for the search form.

A search form is made up of the code that creates the user interface and the code that generates the search query.

Select from the following search form tasks:

Working with a search form

- Creating_a_basic_search_form
- Creating_a_multi-field_search_form
- Using_the_Java_Word_Wheel in a search form

Manipulating the search query

- Submitting_a_search_string to the server
- Limiting_the_scope_of_a_search

Adding a search form to a site

- Registering_a_search_form with NXT 3
- Displaying_a_search_form_list in NXT 3

Creating a Basic Search Form

Site_Design_Introduction

A basic search form uses a single "text" (or "textarea") element and a button that submits the search to the NXT 3 Server.

A basic search form is included in the banner.xsl file in your default templates directory. You can modify the file, copy the entire file to another file, or copy just the applicable form code to create your own basic search form.

To create a basic search form, perform the following steps:

1. Create a search form page for your site.
2. Type the HTML code that includes search fields and buttons in the file. You can set search options for the user, or let the user set their own options (although this would be closer to an advanced search form, depending on how much control you give the user).
3. Type the hidden fields that help generate the search query, or pass them through the URL. See Requesting a Search Results Page in *Requesting Content from NXT 3 Help*.
4. Save the file.

See Also

Registering_a_Search_Form_with_NXT_3

Search_Form_Concept

Search_Form_Design_Concept

Search_Form_Templates_Concept

Search_Results_Parameters in *Requesting Content from NXT 3 Help*

Creating a Multi-Field Search Form

Site_Design_Introduction

A multi-field search form provides several input areas. These are often used to search specific fields in a content collection or to prompt the user to enter certain types of information. These search forms let the user set certain values.

A multi-field search form requires scripting. The information entered into each input element must be combined appropriately to produce a valid search string. The search form's "xhitlist_q" element string must then be set equal to the search string. Additional scripting may also be necessary to set the search options.

See Also

Registering_a_Search_Form_with_NXT_3

Search_Form_Concept

Search_Form_Design_Concept

Search_Form_Templates_Concept

Creating_a_Multi-Field_Search_Form_Example

last-modified_Field_Reference

locked_Field_Reference

locked-by_Field_Reference

Search_Results_Parameters in Requesting Content from NXT 3 Help

Using the Java Word Wheel in a Search Form

Site_Design_Introduction

Searching fielded information or headings can be tricky for users. Unless you provide a static list of options, the users must guess when deciding which term might appear in the field or heading they are searching. Adding a word wheel to a search form gives the users a list of terms to select from based on the search form's criteria. NXT 3 includes a Java_applet_for_a_word_wheel function.

To use the Java applet in a search form, do the following:

1. Create a new search form, or modify an existing one.
2. Type the HTML and JavaScript code for inserting the applet, including the JavaScript hooks and the applet query parameters.
3. Type the HTML and JavaScript code for using the edit box, including the JavaScript hooks and the applet parameters.
4. Save the file.

See Also

Search_Form_Concept

Search_Form_Design_Concept

Search_Form_Templates_Concept

Using_the_Java_Word_Wheel_Example

Submitting a Search String to the Server

Site_Design_Introduction

A search form uses the "xhitlist_q" parameter to specify a NXT 3 query string. Search options are set through the other arguments used to Request a Search Results Page. Use form elements to allow the user to specify search terms and search options. Use the form's "ONSUBMIT" script to retrieve information from a form's elements and set it into the forms parameters for compilation into a URL.

For example, a search form might include a "test" element named "Query" to allow the user to enter a query string. When the user submits the form, a script assigns the form's "xhitlist_q" parameter the value specified in the Query element. For multi-field search forms, you must use a script to gather the search terms from the various input elements and create a single query string stored in the "xhitlist_q" parameter.

You can also use option buttons to switch between the available query syntax methods. Depending on the option button selected, the "ONSUBMIT" script sets the "xhitlist_x" parameter to the correct syntax.

The sample script below shows how to assign the contents of the "Query" element to the "xhitlist_q" parameter. It also shows how to set the query syntax. The script assumes that it is called when the "onSubmit" event is triggered.

```
<form name="query" action="#!-- #EXECUTIVE:SCRIPT_NAME --#" method="GET">
 <input type="text" size="15" name="xhitlist_q"/>
 <input type="submit" class="button" value="Search"/>
 <input type="hidden" size="0" name="f" value="xhitlist" />
 <input type="hidden" size="0" name="xhitlist_x" value="Simple" />
 <input type="hidden" size="0" name="xhitlist_s" value="relevance-weight"/>
 <input type="hidden" size="0" name="xhitlist_d" value=""/>
 <input type="hidden" size="0" name="xhitlist_hc" value=""/>
 <input type="hidden" size="0" name="xhitlist_xsl" value="xhitlist.xsl"/>
 <input type="hidden" size="0" name="xhitlist_vpc" value="first"/>
 <input type="hidden" size="0" name="xhitlist_sel"
  value="title;path;relevance-weight;content-type;home-title" />
</form>
```

See quickfactssearchexample.htm in your NXT 3 templates directory for more detailed examples.

Limiting the Scope of a Search

Site_Design_Introduction

It is possible to limit the scope of a search through the search form. When a user submits a query using this scope-limiting search form, results display for only the matches that occur in the specified search scope. The scope is defined by domain.

You can find additional information about limiting the scope of a search in Narrowing a Search to Specified Branches of a Site, in *Requesting Content from NXT 3 Help*, and Changing the Scope of a Search, in *Client Query Syntax Help*.

See Also

last-modified_Field_Reference

locked_Field_Reference

locked-by_Field_Reference

Registering a Search Form with NXT 3

Site_Design_Introduction

To have your search form accessible by the Search Form component, register it with the NXT 3 Server. Although it is possible to use a search form without registering it with the server (by linking to it as you would any other HTML file), registering the search form lets you use the Search Form component to display your search form in a dynamic list of other registered search forms.

To register the search form, add it to the site's Search Form list. Follow the steps in Adding a Search Form, in *Content Network Manager Help*.

Displaying the Search Form List

Site_Design_Introduction

To have a list of search forms displayed in the site, do the following:

1. Create a file that will display the list of search forms, or edit an existing file.
2. Type the search forms replacement variable (`#!-- #searchforms:select --#`) in the appropriate place.

   The default NXT 3 site displays the list in a drop-down menu, which is created from the banner.xsl file.

   The code for this is similar to the following:

   ```
   <form method="get">
   <select size="1">
   <option value="#NoForm">Select Search Form</option>
   #!-- #searchforms:select --#
   </select>
   </form>
   ```
3. Save the file.

Site Design Search Results Tasks

Site_Design_Introduction

If you allow your users to search your NXT 3 site, you need to have a search results page that displays the matches for user searches.

The following tasks can help you as you create or modify a search results page:

- Creating a search results page.
- Changing the viewport on the search results page.
- Showing the last search query in the search form.
- Moving between search matches.

Creating a Search Results Page

Site_Design_Introduction

The default NXT 3 site uses an XML-based search results page, but it is also possible to use an HTML search results page using the HITLIST component.

XML Search Results Page

To make use of the XML search results page, simply make the search request (in the form and query syntax) to the XHITLIST component and supply the name of the stylesheet that needs to be applied for formatting the results. For example: `<input type="hidden" size="0" name="f" value="xhitlist" />` points to the XHITLIST component, and `<input type="hidden" size="0" name="xhitlist_xsl" value="xhitlist.xsl"/>` tells the component what stylesheet to use.

To modify how the XML search results page looks, open and edit the xhitlist.xsl file in your NXT 3 templates directory.

> **Note:** We recommend that you create a copy of the existing xhitlist.xsl file, rename the copy, and use that renamed copy when pointing to the stylesheet to use from the search form so that you can make changes without affecting the original.

HTML Search Results Page

To create an HTML search results page, do the following:

1. Create a new file for the search results page, or edit an existing one.
2. In the body of the file, type the replacement variable for the HITLIST component: `<!-- #HITLIST?v=2.0 -->`.
3. Save the file.

See Also

Requesting_a_Search_Results_Page

last-modified_Field_Reference

locked_Field_Reference

locked-by_Field_Reference

Changing the Viewport on the Search Results Page

Site_Design_Introduction

When a user performs a search that has many matches, you may only want a certain number of matches to display on the search results page at a time so that the search results page does not take a long amount of time to load. You can enable "viewports" that show a given number of results per page, and have links to previous and next results pages with additional matches.

The xhitlist.xsl file in the default templates directory has an example of how to make this work.

To add a link to the previous and next viewports, do the following:

1. Open your site's copy of xhitlist.xsl in an editor.
2. Add a (or modify the existing) template block to see if the current viewport is the last available viewport, using the "view-is-end" element.

   For example, `<xsl:if test="list-section/view-is-end[.='no']">`.
3. Type the appropriate HTML code for a table row and cells. The number of cells would vary depending on the number of columns in the table. The default search results page has six columns.
4. In the table cell, type the link code for the next viewport. For example, `<a><xsl:attribute name="href">#!-- #executive:script_name --#?f=xhitlist$xhitlist_vpc=next</xsl:attribute>More results</a>`.
5. Add a (or modify the existing) template block to see if the current viewport is the first viewport, using the "view-is-begin" element.

   For example, `<xsl:if test="list-section/view-is-begin[.='no']">`.
6. Type the appropriate HTML code for a table row and cells. The number of cells would vary depending on the number of columns in the table. The default search results page has six columns.
7. In the table cell, type the link code for the previous viewport. For example, `<a><xsl:attribute name="href">#!-- #executive:script_name --#?f=xhitlist$xhitlist_vpc=prev</xsl:attribute>Previous results</a>`.
8. Save the file.

Showing the Most Recent Search Query

Site_Design_Introduction

It may sometimes be appropriate to display the most recent search query in the search results page, or when the user returns to the search form.

To add the search query to the search results page, do the following steps:

1. Open your site's copy of xhitlist.xsl in an editor.
2. Add the code for an HTML table above the search results list table section (or add a table row to the existing table).
3. In the table cell, type the code for displaying the search query. For example, `<xsl:value-of select="list-section/query"/>`.
4. Save the file.

Note: To have the previous query show up in the HTML search form, enter the HITLIST component replacement variable in the "q" parameter. For example: `#!--#hitlist:query --#`.

See Also

Saving_Search_Queries_Example

Navigating Between Search Matches

Site_Design_Introduction

NXT 3 supports navigating between search matches in an HTML document by inserting anchors into the HTML at match locations. Anchors are named "LPHit + <Hit Number>". For example, the following anchor is inserted at the location of the first match in a document: `<A NAME="LPHit1">`. Attach JavaScript code to a document navigation item to move between hits. See the "prevMatch" and "nextMatch" functions in the document-tools.htm template for an example of JavaScript that does this. The scripts use JavaScript to keep track of which match is the current match (see the hit-tracker.js file for more information).

Site Design Security Tasks

Site_Design_Introduction

NXT 3 offers you the ability to secure access to the site.

The following tasks guide you in setting up (or disabling) security for your NXT 3 site:

- Setting_up_site_security.
- Changing_site_security settings.
- Modifying_the_security_settings_for_a_user.

Setting Up Site Security

Site_Design_Introduction

Access control is provided to secure the Content Network Manager application and control access to content on the server. NXT 3 provides a simple interface for access control through the use of an INI file. More complex access control implementations can be created using the LDAP service. Information on using access control is provided in the Access_Control_Help document included with NXT 3.

After creating a new site, you need to either disable_access_control for the NXT 3 Server, or configure_the_access_control_module_for_the_new_site_and_view.

See Also

Configuring_ACM

Configuring_DefaultACM

Turning_On_Access_Control

Changing Security Settings for a Site

Site_Design_Introduction

To change the security settings for a site, select from the following options that are detailed in *Content Network Manager Help*:

- Configuring_ACM
- Configuring_DefaultACM
- Turning_Off_Access_Control
- Turning_On_Access_Control

Changing User Security Settings

Site_Design_Introduction

To change security settings for a specific user, do the following:

1. Open Content Network Manager for the NXT 3 Server that you want to change a user's security settings.
2. Select the server listed in the left-hand pane.
3. From the **Action** menu, choose **Properties**.

   or

   Click  on the toolbar.
4. Click the **Properties** button for the **Access control module** field.
5. Click the **Edit Users** button to open the Access Control Users dialog box.
6. Click the **Username** for the user who needs to have settings changed (or click the button to add a new user).
7. Click the **Access property** that you need to change and click **Edit**.

   For instance, to change the "View-id" setting for a user (allowing or disallowing a user to access a certain view), select **Edit** and add or remove a view ID from the list.
8. Type a new value, or change the current one.
9. Click **OK** to close the dialog box.
10. Repeat as necessary.
11. Click **Close**, then **OK** twice to return to the Content Network Manager main window.

See Also

Access_Control_Help

Configuring_ACM

Configuring_DefaultACM

Turning_On_Access_Control

Site Design Site Tasks

Site_Design_Introduction

Each NXT 3 Server can host multiple sites. NXT 3 comes with a standard site, called "NXT," that uses the templates that also come with the product.

You can change this default site and its templates, or you can create a new site with or without new templates.

Use the following tasks as you work with your NXT 3 site:

- Creating a new_site.
- Creating a new_site_using_new_templates.
- Creating a new_site_with_modified_templates.
- Adding_content to a site.
- Adding searchable_fields to content.

Creating a New Site

Site_Design_Introduction

When creating a new site, you need to choose from one of three options:

- Create a new site using the same templates as the default site, using the steps from *Content Network Manager Help*.
- Create a new site and modify the existing templates.
- Create a new site with new templates.

While the steps for creating the new site are basically the same for all three options, preparation and time-commitment differ dramatically.

You can also create multiple sites on one NXT 3 Server.

See Also

Creating_Multiple_Sites_Example

Creating a New Site with New Templates

Site_Design_Introduction

Use these steps when you want to create an entirely unique and new site with your own templates.

1. Since you don't want to interfere with the default templates, we recommend that you create a new directory for your new templates on the hard drive of the computer that is the server for the new site. For instance, C:\newsite.

   Note: You may want to create additional directories within the new directory for the template files, images, and stylesheets since this new directory will probably contain content for the site as well.

2. Create new templates in the new templates directory.
3. Complete steps 1-4 from Adding a New Site in *Content Network Manager Help*.
4. On the **General** tab of the Site Properties dialog box, change the **Default Page** file information for the Web page that you want to have open when a user accesses the site. Make sure that this file is located in the appropriate templates directory for this site.
5. Select the **Documents** tab. If you created a new templates directory, type the appropriate base path for **Templates Path**, **Image Path**, and **Stylesheet Path** in the fields provided.

   Note: Typically, all the paths point to the same directory. For instance, C:\newsite.

6. Select the **Views** tab.

   If you want to add a new view for this site, follow steps 5-8 for adding a new view in *Content Network Manager Help*.

   If you want to use the default view and you created a templates directory that is different from the default directory, you need to change the **Template Path**, **Image Path**, and **Stylesheet Path** to correspond to the new paths where these files are located. Select the view you want to edit, and then click **Edit**.

   Note: These paths are relative to the base path you entered on the **Documents** tab. If these files are in the main directory, make all these fields blank.

   Note: If **Browse** does not work, you may need to delete the default values and then click **Browse** again to find the correct directory.

7. Click **OK** to get back to the Content Network Manager main window.
8. From the **Console** menu, choose **Apply All Changes**.

   or

   Click  on the toolbar.

You now have a new site with new templates.

Before you or a user can view the site, you need to do the following:

1. Add content to the site, including any folders, content collections, content network links, and content services that you want.
2. Optionally, restrict access to specific content for the view.
3. Edit access control properties for users so a user can access the new site and its view.
4. Add the view ID element to the end of the URL for your default site, so it looks something like this:

   For Windows servers:

   http://localhost/NXT/gateway.dll?f=templates$fn=<YourDefaultPage.htm>$vid=<YourSiteName>:<YourViewID>

   For Unix servers:

   http://localhost/NXT/gateway.so?f=templates$fn=<YourDefaultPage.htm>$vid=<YourSiteName>:<YourViewID>

Creating a New Site with Modified Templates

Site_Design_Introduction

When you create a new site, but do not want to put forth the extra effort to create completely new templates, and instead want to modify the existing templates, do the following:

1. Since you probably don't want the default site templates to be just like the new site's templates, we recommend that you create a new directory for the modified templates on the hard drive of the computer that is the server for the new site. For instance, C:\newsite.

   Note: You may want to create additional directories within the new directory for the template files, images, and stylesheets since this new directory will probably contain content for the site as well.
2. Copy the default template files to the new templates directory.
3. Complete steps 1-4 from Adding a New Site in *Content Network Manager Help*.
4. On the **General** tab of the Site Properties dialog box, if you want, you can change the **Default Page** file information for the Web page that you want to have open when a user accesses the site. Make sure that this file is located in the appropriate templates directory for this site.
5. Click the **Documents** tab. If you created a new templates directory, enter the appropriate base path for **Templates Path**, **Image Path**, and **Stylesheet Path** in the fields provided.

   Note: Typically, all the paths point to the same directory. For instance, C:\newsite.
6. Select the **Views** tab.

   If you want to add a new view for this site, follow steps 5-8 for adding a new view in *Content Network Manager Help*.

   If you want to use the default view and you created a templates directory that is different from the default directory, you need to change the **Template Path**, **Image Path**, and **Stylesheet Path** to correspond to the new paths where these files are located. Select the view you want to edit, and then click **Edit**.

   Note: These paths are relative to the base path entered on the **"Documents** tab. If these files are in the main directory, make all these fields blank.

   Note: If **Browse** does not work, you may need to delete the default values and then click **Browse** again to find the correct directory.
7. Click **OK** to get back to the Content Network Manager main window.
8. From the **Console** menu, choose **Apply All Changes**.

   or

   Click  on the toolbar.

You now have a new site with the modified templates.

Before you, or a user, can view the site, you need to do the following:

1. Add content to the site, including any folders, content collections, content network links, and content services that you want.
2. Optionally, restrict access to specific content for the view.
3. Edit access control properties for users so a user can access the new site and its view.
4. Add the view ID element to the end of the URL for your default site, so it looks something like this:

   For Windows servers:

   http://localhost/NXT/gateway.dll?f=templates$fn=<YourDefaultPage.htm>$vid=<YourSiteName>:<YourViewID>

   For Unix servers:

   http://localhost/NXT/gateway.so?f=templates$fn=<YourDefaultPage.htm>$vid=<YourSiteName>:<YourViewID>

Adding Content to a Site

Site_Design_Introduction

You can add folders, content_collections, content_network_links, and content_services to a site.

Content Network Manager Help provides instructions on how to add the following types of content to a site:

- Add a folder.
- Add a blank_content_collection.
- Add a content_collection_with_existing_content.
- Add a content_network_link.
- Add an ODBC_database_service.
- Add an Oracle_database_service.
- Add a file_system_service.
- Add a Web_site_service.

Adding Searchable Fields to Content

Site_Design_Introduction

Many documents already have some searchable fields, or metadata, associated with them. These can be tags or direct metadata references, as in HTML pages.

You can also add fields to your XML content that users can search on. To add a searchable field to your XML content, do the following:

1. Open the content in its native application.

   Note: You can also add some metadata (for field searching) to content by using Manage Content to edit the content's properties. You can edit the **Title**, **Subject**, **Author**, and **Abstract** fields in this manner.
2. Add a unique set of tags around the desired text, or note the tags that already surround the text.

   For instance, for a company directory, you can put "employee" tags around everyone's names: `<employee>Fred Employee</employee>`.
3. Save the file.
4. Repeat steps 1-3 for all the content that you want to have this searchable field.
5. If content is in a content collection that is built using a makefile, add the fielding tags to the makefile.
6. Open the indexsheet file (usually a file with the XIL extension) for the content type, or create a new one.
7. Define the fields inside the "np:definitions" element, using the following form:

   <field name="<FieldName>" type="text"/>

   <FieldName> equals the name of the tag that you created. In the example above, you would replace <FieldName> with *employee* to get the following: `<field name="employee" type="text"/>`.
8. Before or after the existing template match sections, add a template match for your field, as follows:

   ```
   <xsl:template match="<FieldName>">
   <np:index field="<FieldName>">
   <xsl:process-children/>
   </np:index>
   </xsl:template>
   ```
9. Save the indexsheet.
10. Edit the content collection makefile, or the content properties to use the new or changed indexsheet.
11. Rebuild the content and save the site changes.
12. Search for content in the field by entering the following search phrase into the Boolean Search form:

[field <FieldName>:<SearchTerms>]

Site Design Template Tasks

Site_Design_Introduction

Template tasks can help you as you work with the framesets and templates supplied with NXT 3.

In this section, the following tasks direct you in working with templates and framesets:

Frameset Tasks

- Using frames in a site.
- Specifying the site's main page.
- Creating a table of contents frame and template.
- Creating a document frame and template.

Template File Tasks

- Working with templates.
- Creating a document navigation template.
- Modifying the document navigation template.

Using Frames for a Site

Site_Design_Introduction

The default site for NXT 3 uses frames for displaying data in the table_of_contents, toolbar, and document frames.

Frames are defined through a frameset tag inside an HTML document. You can use as many frames as you like in your site. However, you need to keep track of the frame names so that linking and loading documents in a frame work well.

The frames for the default NXT 3 site look similar to the following table:

<table>
<tr><td colspan="2">banner.xsl</td></tr>
<tr><td>contents-frame-h.htm (which loads contents-h.htm) OR contents-frame-j.htm</td><td>document-frameset.htm creates frameset rows in this space that include document-tools.htm on the top 35 pixels, and document-frame.htm on the bottom, which loads the appropriate documents.</td></tr>
</table>

If you change the names of the default documents, you need to change the reference in the frameset file as well.

Specifying the Site's Main Page

Site Design Introduction

The main page (sometimes referred to as the "default page") of a site is the page that comes up first when a user comes to a web site. For NXT 3, the main page of the default site is TitlePage.htm. This is the page that appears in the document frame when someone first accesses the default site. Since NXT 3 uses frames, the master frames page, default.htm, references TitlePage.htm.

If you enter the URL of the site as http://localhost/NXT/gateway.dll (for Windows) or http://localhost/NXT/gateway.so (for Unix), you'll notice that the default page for the site really is TitlePage.htm. Adding `?f=templates$fn=default.htm` to the end of the URL makes the site load the default.htm file first.

To specify the default page for a site, do the following:

1. Perform steps 1-3 from Editing Site Properties, in *Content Network Manager Help*.
2. In the **Default Page** field, click **Browse** to select the default page, or type the name of the file in the field provided.

   Note: The file you specify should be in the site's templates directory.
3. Click **OK** to accept the changes.
4. Save the changes to the server.

Creating a Table of Contents Template

Site Design Introduction

The table of contents displays the structure of the site. You can create a table of contents template file that dynamically updates when content and structure changes (rather than creating a static table of contents file that you would need to manually update).

To create a table of contents template file, do the following:

1. Create a table of contents file, or use an existing one.
2. Open the table of contents file in an editor.
3. If you want an HTML table of contents, call the Contents component by putting a replacement variable in the body of the file. For example, `<!-- #CONTENTS -->`.

   If you want a Java table of contents, enter the appropriate **applet code** in the body of the file.
4. Create a new template file called contents-h.htm, or use contents-h.htm in the templates directory for the default site as a guide.
5. Open the file in an editor.
6. Type the replacement variable `<!-- #CONTENTS:parents -->` to display any nodes above the user's current one. Type `<!-- #CONTENTS:children -->` to display the node titles of any node beneath the current one.
7. Save the file.

Creating a Document Template

Site_Design_Introduction

The document template calls the Document component to display the appropriate contents.

To create a document template, do the following:

1. Create a new file called document_frameset.htm (or use the one in the template directory of the new site).
2. Open the template file in an editor.
3. Add a replacement variable to the file that calls the Document component. For example: `<!-- #DOCUMENT?t=main.htm -->`, where main.htm is the name of the template file.
4. Save the file.

You may also need to create a template file for displaying documents that are HTML fragment types. To do this, follow these steps:

1. Create a new file called document.htm (or use the one in the template directory of the new site).
2. Add a replacement variable to the body of the file (between the HTML "body" tags) that streams in the fragment file's contents. For example: `<!-- #DOCUMENT:t=document.htm -->`, where document.htm is the name of the template file.
3. Save the file.

Working with Template Files

Site_Design_Introduction

NXT 3 includes many types of template files that you can use or customize however you want.

The following template files are required for the NXT 3 site to work:

- contents-h.htm
- document.htm
- document-frameset.htm

All other files can be the same as the ones used for the default site, or ones you create yourself.

You also need a default page for the site, frameset pages to load the various templates, and access control files (challenge.htm and denied.htm are the default ones) if you implement access control.

See Also

Templates_Concept

Creating a Document Navigation Template

Site_Design_Introduction

The document navigation template allows a user to switch between a document, search results, the document and search results, and a search form.

The default NXT 3 site uses the metaphor of tabs to allow users to do this. For the default site, banner.xsl contains the code that creates the tabs and loads the appropriate content in the document frame.

To create the document navigation template, do the following:

1. Create a new template file, or copy banner.xsl.
2. Open the file in an editor.
3. Type the code to insert the tab graphics in the page. For example, `<img src="#!-- #IMAGES:image.gif --#"/>`.
4. Type the link code that updates the document frame with the appropriate content. For example:

   For the document tab, type `<a href="#!-- #TEMPLATES:document-frame-set.htm --#" target="main">`.

   For the results tab, type `<a href="#!-- #EXECUTIVE:SCRIPT_NAME --#?f=xhitlist$xhitlist_xsld=xhitlist.xsl" target="main">`.

   For the split results and document tab, type `<a href="#!-- #TEMPLATES:doc-results.htm --#" target="main">`.

   For the search form tab, type `<a href="javascript:showLastSearch-Form()">`, where you have a JavaScript function called "showLastSearchForm" that tracks which search form was used last to display in the frame. Optionally, you can type code that always loads the same search form.

Modifying the Document Navigation Template

Site_Design_Introduction

The document navigation template is located in banner.xsl. Some other document navigation functionality is also available in document-tools.htm.

Before modifying anything in the files, NextPage recommends that you create a backup copy of the templates files.

The banner.xsl template contains code for creating the navigation tabs, as well as some search functions and Manage Content.

The document-tools.htm template contains code for synchronizing the table of contents with the displayed document, moving to the next and previous search matches, clearing the highlighting created by a search match, finding similar search results to the selected document, and showing the reference information about the current document.

See Manage_Content_Help for more information about document navigation functionality.

To remove a function from the templates, simply delete the code for that function. For instance, to remove user's ability to use the Manage Content function, delete the code that creates the link and image. You could also delete or change the Manage Content JavaScript code so that when the link is clicked, the user gets a message saying that the function does not work or is not available (you do this by deleting the code from the function, and adding code to create a message, for example: `alert("Message here");`).

To add functions to the template, insert the appropriate link in the body of the document. You may also need to add a JavaScript function that is processed when a user selects the link.

Site Design Table of Contents Tasks

Site_Design_Introduction

With NXT 3, you can create a table of contents to help your users navigate the site.

The following tasks deal with the table of contents:

- Creating an HTML_table_of_contents.
- Creating a Java_table_of_contents.
- Syncing_the_table_of_contents with documents.

Creating an HTML Table of Contents

Site Design Introduction

The HTML table of contents gives the user an easy way to explore the content of the site. The HTML table of contents is a good option if you are unsure about what types of web browsers your users use to access the site. Otherwise, the Java table of contents provides some additional functions as long as your users' browsers support a Java applet.

To create an HTML table of contents, follow these steps:

1. If you are using frames, make sure the main frameset points to the page that contains the table of contents.
2. Create the file that is referenced in the frameset, or edit an existing file.
3. Add a replacement variable to the file that calls the Contents component. For example, `<!-- #CONTENTS -->`.
4. Open the contents-h.htm file.

   Note: This file is required if you want to have an HTML table of contents.
5. Add replacement variables to the file that display the parent and children nodes. For example, `<!-- #CONTENTS:parents -->` and `<!-- #CONTENTS:children -->`.
6. Save the files.

See Also

Requesting an HTML Table of Contents Page in *Requesting Content from NXT 3*

Creating a Java Table of Contents

Site_Design_Introduction

The Java table of contents gives the user an easy way to explore the content of the site. Since the Java table of contents is an applet, the table of contents only has to load once and the page does not need to be refreshed. It provides some added functions over the HTML_table_of_contents, such as being able to view multiple branches of the site at once, and faster response to user requests.

> **Note:** When a user who has a computer with Windows XP and Internet Explorer 6, and then first accesses the Java table of contents, the Windows XP Install-On-Demand program asks the user if the user wants to trust Microsoft and download the Java Virtual Machine (JVM). Users should select "Yes." After the download completes, the Java table of contents works as normal. The user does not need to worry about installing JVM again on that machine.

To create a Java table of contents, do the following:

1. If you are using frames for your site, edit the frameset to point to the Java table of contents file.
2. In the table of contents page, enter the code_for_the_applet and the parameters you want for it.
3. Save the file.

See Also

Requesting_a_Java_Table_of_Contents_Page in *Requesting Content from NXT 3 Help*

Syncing the Table of Contents with a Document

Site_Design_Introduction

The default NXT 3 site does not automatically sync the table of contents with the current document. That function is provided through a **Sync Contents** button that the user can select.

You can also use this function, or you can set up NXT 3 to always sync the table of contents with the selected document.

Note: Enabling the automatic sync option can cause a decrease in site performance.

To set up automatic syncing, do the following:

1. Open the template file that calls the Contents component.
2. Add the **sync** parameter to the replacement variable. For instance, `<!-- #CONTENTS?sync=1 -->`.
3. Save the file.

Site Design View Tasks

Site_Design_Introduction

A view for a site determines what users can see, and how they see it.

The following tasks provide help in working with site views:

- Creating a view for a site.
- Modifying a view for a site.

Creating a New View

Site_Design_Introduction

A site can have one or more views associated with it.

To create a new view, do the following:

1. Complete steps 1-6 from Adding a View in *Content Network Manager Help*.
2. If you are using the default templates directory that came with NXT 3, skip to step 3, otherwise change the **Template Path**, **Image Path**, and **Stylesheet Path** to correspond to the appropriate paths.

   Note: These paths are relative to the base path entered on the **Documents** tab. If these files are in the main directory, make all these fields blank.

   Note: If **Browse** does not work, you may need to delete the default values and then click **Browse** again to find the correct directory.
3. If you want to limit user access to specific content in this view, click **Define Domain**. When you are done, click **OK** to save the changes and exit, or **Cancel** to ignore any changes and exit.

   Note: To select only a portion of the site, you must first clear the topmost node check box for the site. Then, you can select individual folders and content, down to a document level.
4. Select **OK** twice to return to the Content Network Manager main window.

To access this view, you must add it to the View-id of the users who need to see it.

Modifying a View

Site_Design_Introduction

To modify the properties of a view, see Editing a View in *Content Network Manager Help*.

To change the content that user can see in a view, follow the steps in Defining a Domain in *Content Network Manager Help*.

Site Design Concepts

Concepts provide background information on specific portions of the product. Links to concepts may be found throughout Help topics.

Site Design Help covers the following concepts:

- Site design information
- Design of a search form
- Templates
- Document navigation template
- Frameset templates
- Search form template
- Global parameters
- Hosting multiple sites
- Search form
- Java word wheel
- Searchable field
- View

Site Design Concept

Site Caching

When designing a site, you want to be able to view the results of your changes quickly. The NXT 3 installation configures the server so that templates are cached in memory. Each time you modify a template you must bring down the Web server to force NXT 3 to reread the template from disk so you can test the results of your modifications.

To turn off caching, modify the siteview.ini file in the bin directory of your NXT 3 installation. In the [FILECACHE] section, specify `Cache=0`. Specifying this option causes NXT 3 to reread templates from disk any time they are changed. Because NXT 3 never frees templates from memory, you should restart your Web server after a significant number of changes to your templates.

When you are done designing the site, change siteview.ini to the originial `Cache=1` to allow for caching.

Replacement Variables

NXT 3 makes extensive use of Replacement Variables throughout the templates. Replacement variables are expressed as HTML comments using either <!-- and --> or #!-- and --#. They both do the same thing, so which form you use is merely a matter of personal preference.

See Requesting Content from NXT 3 Help for more information.

Search Form Design Concept

The following suggestions can assist you as you plan the number and type of search forms to provide at your site. The suggestions are not presented in any particular order.

Provide a simple form for simple searches. This may appear obvious, but the tendency is to add too much to a form. For quick searches, a text input field and a submit button are often enough.

Provide a multi-field form for "advanced" or "detailed" searches. For more complicated searches, or for instances where you want to guide a user through the search process, using several text input fields is required.

Use the "getDomain" function. Include the **getDomain** function in a search form to allow the users to limit the scope of their search to selected branches of the site's table of contents. See the "Advanced Search" template included with NXT 3 (AdvSearch.htm) for an example on calling the function. The **getDomain** function is found in the SearchForm.htm template.

Use the domain element (d) to limit the scope of a search. Hard code this value in templates designed for a specific content collection or a set of content collections.

Turn off caching when developing a search form. When developing search forms you want to be able to view the results of your changes quickly. The NXT 3 install configures the server so that templates are cached in memory. Each time you modify a template you must bring down the Web server to force NXT 3 to reread the template from disk so you can test the results of your modifications. To turn off caching, modify the siteview.ini file to specify `Cache=0` in the [FILECACHE] section. Siteview.ini is a configuration file located in the NXT 3 bin directory. Specifying this option causes NXT 3 to reread templates from disk any time they are changed. Because NXT 3 never frees templates from memory, you should restart your Web server after a significant number of changes to your templates.

Templates Concept

NXT 3 uses templates to create the site interface. Templates make use of NXT 3s replacement variables to include content dynamically in the site. They also allow you to stream documents and the contents of those documents into the template files. See Shipping Templates Reference for more detailed information about the templates and how they are used.

You can modify many of these template files to create a unique look for your site.

Document Navigation Template Concept

A document navigation template is an HTML page containing links that execute actions such as moving between documents or moving between documents with matches. This information explains the JavaScript and NXT 3 URLs typically used by document navigation templates.

The following example shows how you can specify a template for a document navigation frame.

```
<!--  We'll use script to write out the frameset
so that we can pass the hash used to request this page through
to the actual document. The hash is used to
link to the first search match, to do PDF highlighting,
and to link to a subdocument table of contents entry. This
only works on the first hash, though, since IE doesn't reload the
document if only the hash on the new URL changes. Subsequent hashes
need to notify this window by calling the changeHash() function
above. -->

<script type="text/javascript">

document.write('<frameset rows="35,*" border="0" frameborder="0">
document.write('<frame name="doc-tools"
src="#!-- #TEMPLATES:document-tools.htm --#
"noresize="noresize" scrolling="no"
marginwidth="5" marginheight="0"/>');
document.write('<frame name="doc-body"
src="#!-- #TEMPLATES:document-frame.htm --#
$q=#!-- #EXECUTIVE:PARAM:q --#$x=#!-- #EXECUTIVE:PARAM:x --#'
+ location.hash + '" noresize="noresize"/>');
document.write('</frameset>');
initPage();
</script>
```

When NXT 3 parses the `<!-- #TEMPLATES:... -->` HTML comment, it replaces it with the path to the templates directory of the current view. `#Templates` is one of the NXT 3 Replacement Variables. Specify a template variable in a template file by including it in an HTML comment and preceding it by the pound character (#).

This example assigns the document navigation frame the name "document-tools." This allows the other frame pages to reference it.

Frames Concept

When you create a Web site, you can use frames to define multiple, independently controllable sections. This effect is achieved by building each section as a separate HTML file and having one *master* HTML file identify all of the sections. When a user requests a web page that uses frames, the address requested is actually that of the master file that defines the frames; the result of the request is that multiple HTML files are returned, one for each visual section. Links in one frame can request another file that will appear in another (or the same) frame.

The NXT 3 default site makes use of a frameset so that the navigation and tool options are always visible to the user throughout the entire site. Within each frameset, one or more frames are defined.

See Making the Initial Request in *Requesting Content from NXT 3 Help* for information on the URL to use to request a frameset's master page.

Search Form Template Concept

A search form template is an HTML file containing a FORM element used to submit a search to NXT 3. Submitting an HTML form results in a request for a URL. The form's parameters are concatenated onto the end of the URL. A search form submits a URL that requests a search results page and specifies that the URL be redirected to a given frame set. See Requesting a Search Results Page in *Requesting Content from NXT 3 Help* for information on the URL syntax to use to request a search results page. The following example shows the SRC tag used to call the banner.xsl page, which displays the search form list:

```
<frameset rows="80,*" border="0"
onload="initPage()">
<frame
name="banner" src="#!-- #executive:script_name --
#?f=userinfo&userinfo_xsl=banner.xsl&userinfo_cat
=saved-search" scrolling="no"
noresize="noresize" marginwidth="0"
marginheight="0">
<frameset cols="217,*">
<frame name="contents" src="#!--
#TEMPLATES:contents-frame-h.htm
--#&sel=0&tf=main&tt=document-frameset.htm&t=
contents-frame-h.htm&och=onClick">
<frame name="main" src="#!--
#TEMPLATES:document-frameset.htm
--#">
</frameset>
<noframes>
<body>

<p>This site requires a frames-capable browser.</p>

</body>
</noframes>
</frameset>
```

When the previous frame example is requested, the banner.xsl page is called and the contents of the page is placed within the frame. The banner.xsl page provides a form for performing a Simple Search as well as showing a list of available search forms. The following example shows the "select" tag and replacement variable used to display search forms from within the banner.xsl page:

```
<select name="searchform_list"
onchange="showSearchForm(this.options
[this.selectedIndex].value);
this.selectedIndex = 0" >
```

```
<option value="#NoSelection">Choose
Search Form</option>
<!-- #SEARCHFORMS:select -->
</select>
```

The `<!-- #SEARCHFORMS:SELECT -->` variable lists the search forms available to the site. When the page containing this variable displays, the variable is replaced with the actual search form names. The following shows a portion of the source of the page containing the `<!-- #SEARCHFORMS:SELECT -->` variable:

```
<select name="searchform_list"
onchange="showSearchForm(this.options
[this.selectedIndex].value);
this.selectedIndex = 0">
<option value="#NoSelection">Choose Search Form>
<OPTION
Value="contents:10.1048%2FAdvancedSearch">Advanced
Search>
<OPTION
Value="contents:10.1048%2FBooleanSearch">Boolean
Search>
<OPTION Value="no-contents:10.1048%2FNXT3%2F3.
0%2FQuickFactsExampleSearchForm">Country
Quick Facts Search>
</select>
```

The `<option value="contents:10.1048%2FAdvancedSearch">` command tells NXT 3 to serve the Advanced Search form. This same command structure is used to display the Boolean Search form (as seen in the next line of the example). A "select" command specifies the page to load when the search form is selected: `<select name="searchform_list" onchange="showSearchForm(this.options[this.selectedIndex].value); this.selectedIndex = o ''>`. For information on creating search forms, see Search Forms Tasks.

Global Parameters Concept

This example shows a URL that creates a category, adds an attribute, and creates three items in the attribute. The URL is split up for ease of viewing.

```
http://my_machine/NXT/gateway.dll?f=userinfo&
userinfo_cat=CATEGORY&
userinfo_c=add&
userinfo_n=NAME1&
userinfo_ipl=PROP1;PROP2;PROP3&
userinfo_PROP1=VAL1&
userinfo_PROP3=VAL3&
userinfo_xsl=STYLESHEET1.xsl
```

The above URL returns the following XML:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="/NXT/gateway.dll?f=stylesheets$fn=STYLESHEET1.xsl$3.0"?>
<CATEGORY>
<item name="NAME">
<PROP1>Val1</PROP1>
<PROP2/>
<PROP3>Val3</PROP3>
</item>
</CATEGORY>
```

A subsequent request is shown as follows:

```
http://my_machine/NXT/gateway.dll?f=userinfo&
userinfo_cat=CATEGORY&
userinfo_c=add&
userinfo_n=NAME2&
userinfo_ipl=PROP1;PROP2&
userinfo_PROP2=MYVAL2&
userinfo_xsl=STYLESHEET2.xsl
```

The second request contributes to the user's XML document and results in the following:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="/NXT/gateway.dll?f=stylesheets$fn=STYLESHEET2.xsl$3.0"?>
<CATEGORY>
<item name="NAME1">
<PROP1>Val1</PROP1>
<PROP2/>
<PROP3>Val3</PROP3>
</item>
<item name="NAME2">
```

```
<PROP1/>
<PROP2>MyVal2</PROP2>
</item>
</CATEGORY>
```

Hosting Multiple Sites Concept

NXT 3 allows hosting of multiple sites. Companies that use NXT 3 on an intranet commonly need to be able to set up various views of their site for different groups of users. In some cases, these sites are used only for organization and not to prevent users from accessing data in another group's site. In other cases, security is an issue and administrators must be able to specify which users can access which site.

The NXT 3 server can host multiple sites, and each site can have multiple views. Each view can potentially allow access to a different subset of a site and present content using a unique interface. The structure of a site and the views accessible from the site are defined within a single site definition file (an XML file).

The following statements briefly describe the functions that multiple site support provides:

- Each site can have its own user interface.
- Each site can share a common set of content collections. In other words, you do not have to maintain a separate copy of a content collection for each site on which it is hosted.
- Each site may restrict access by requiring a user to log in.

A site consists of one or more views of a site that can be accessed from a given Web server alias. A site's set of views can be used to provide different looks or different languages for the site, or both.

All the sites that a NXT 3 server hosts must be registered in a file named SiteView.ini, which is located in the same directory as Executive.dll (typically the NXT 3 bin directory). Each site must have its own section in the SiteView.ini file.

To work with multiple sites, you should understand the following concepts: server, site definition file, site, view, and template.

Search Form Concept

A search form is an HTML form where users enter information that is used to search a NXT 3 site. Simple search forms like the one on the default NXT 3 toolbar contain a single input field for the user to enter search terms into. When the user submits the form, the search terms in the form data is passed to NXT 3 using the POST or GET method. More complex search forms include multiple input fields and use JavaScript to combine input into a search string using NextPage query syntax.

The search string and search options are stored as parameters on the form. When the form is submitted, the values are combined into a URL that requests a search results page (see Requesting a Search Results Page in *Requesting Content from NXT 3 Help* for more information).

Regardless of the type of search form you create, you must:

- Set the Method and Action for the Form
- Manipulate the Search String and Search Options

Once you understand how to do this, creating a basic search form or a multi-field search form is relatively simple.

An HTML form requires that the method and action attributes be set. For search forms, the method should be POST or GET and the action must be `<!-- #EXECUTIVE:SCRIPT_NAME -->`.

POST is a standard method for HTML forms. POST does not place the contents of the search on the URL. For long searches, POST is the recommended method over GET. **GET** places the search contents on the URL to pass it to the server. Since the length of the URL is limited to 2,048 total characters, long searches may not get passed properly on the URL.

`<!-- #EXECUTIVE:SCRIPT_NAME -->` is a NXT 3 template Replacement Variable that resolves to a URL pointing to the NXT 3 Web extension used to process searches.

Submitting a search form results in a URL request for a search results page. See Requesting a Search Results Page and Overview of Searching an NXT 3 Site in *Requesting Content from NXT 3 Help* for information on the syntax used to request a search results page.

See the templates directory of your NXT 3 installation for additional examples. There you will find examples of a tool page with a simple query (banner.xsl), the Boolean Search form (boolsearch.htm) that lets the user enter standard boolean search options, and an Advanced Search Form that shows some advanced options (advsearch.htm).

The QuickFactsSearchExample.htm form shows four forms ranging from a simple search performed on the Country Quick Facts sample content collection to two search forms that use the new Java word wheel. The Quick Facts Search Example form includes comments explaining the sections of code that belong to each search form. This search form is installed with the default templates in the <InstallPath>\Templates\Enu directory.

Java Word Wheel Concept

The Java word wheel provides an interface for users to perform a search based on predefined fields in a set of documents.

The default NXT 3 site includes a Country Quick Facts search form that has a Java word wheel.

The word wheel provides a dynamic list of words from a site and can be navigated by using a scrollbar or by setting focus to the word wheel and using the up, down, page up, page down, home, and end keys. In addition, when it has the focus, the word wheel seeks to the word you type, until a key hasn't been typed for approximately two seconds, at which time it resets the seek function.

The Java "edit box" control is used in conjunction with the Java word wheel. Sometimes, a search form uses multiple edit boxes with only one word wheel. As focus changes to different edit boxes, the word wheel can be made to change to the list specific to the edit box with the current focus.

When you type words in the edit box, the word wheel seeks to the word being typed. You can copy a word from the word wheel into the edit box by double-clicking in the word wheel or by pressing Enter when the focus is in the edit box. Since the edit box does not use the up and down keys, the users are redirected to the word wheel if the focus is in the edit box.

You can implement communication between the word wheel and edit box in several ways. The first and easiest way is to tell the edit box the name of the word wheel to use and let it call the appropriate methods of the word wheel. This is done by setting the word wheel **edit box** parameter to the name of the word wheel applet you would like it to communicate to. Also, set the query **edit box** parameter to correspond to the term list you want displayed in the word wheel when that edit box has focus.

Note: The word wheel cannot display the contents of two fields at the same time.

When the edit box gains focus, it sets the query into the word wheel and moves the word wheel to the first item. It also tells the word wheel to call it back if an item in the list is selected. If multiple edit boxes are using the same word wheel, the list in the word wheel changes each time a new edit box gains focus. Once the edit box has focus, typing or moving the cursor causes the word wheel to seek to the term the cursor is on. Pressing Enter in the edit box or double clicking on the word wheel causes the selected term to replace the term the cursor is on.

If more complicated interaction is desired, you can set JavaScript hook functions and handle the interactions yourself. Each hook is set as a parameter to the applet that overrides the default behavior.

Searchable Fields Concept

NXT 3 normally indexes each word in a document. While this is valuable, it may be necessary to search for specific aspects of the document, like just searching the title, or just documents created by a certain person. An example of this type of search is available in the Document Summary Search Form provided with the default templates.

Many documents already have searchable field information that NXT 3 can use to allow users to search for information in those specific fields. For XML documents, any text between a set of tags can be considered as a field, so users can search on it if they know the tag or if you provide a search form for that tag. These fields are a type of metadata.

Metadata is stored as a document property and describes the document and its children. Most document properties could be considered document metadata. The metadata used by NXT 3 is specific metadata used for searching or resource discovery.

When NXT 3 indexes content, each defined field is given its own search index.

View Concept

A view is a subset of a site's table of contents and optionally a set of templates that provide the interface for the view. A view is defined by inserting a view element in a site definition file. The attributes of the view element specify the view's id, group, template paths, domain, highlight style, and the language for the view. The parent element of the view is considered the root element of the view.

The view's domain attribute specifies which descendants of the root node will be visible in the view. Each view can be given a unique interface by specifying non-default templates and images for the view. In addition, each view can specify the language of the query components to use when searching the view.

Views are created at the server level. Inside a view you can specify a domain that determines the content available for the view. If you define a domain for the root node (the highest level), potentially, everything on the site is available throughout the new view. If you specify that folder A and folder C are part of the domain, then folder B's items are not visible in the new view. The domain allows you to pick and choose the items included in a view instead of entire folders or nodes. Domains can control down to the document and sub-document level (see the steps below for adding domain information).

When selecting items for a domain, you must first clear the highest node's check box in the domain and then select those items that you want to include in the domain. For example, if the highest node is A, you select the check box next to A to remove the check marks from all the check boxes in the domain (A, B, C, and D). If you want specific documents in the D tree, but not B or C tree items, you expand the D tree and select the check boxes of the items in D that you want included. You may notice that A and D both have greyed check boxes with check marks in them. This designates that a portion of the associated tree is selected, not the whole tree structure. If you want all of D's items included in the domain, but not B's or C's items, you need to select the check box for each item in the D tree to include it in the domain.

Site Design Reference

Reference information provides more details for an element within a step of a task. For example, a step requiring inclusion of an ID would have a link from the word ID to reference information titled ID Property Reference. The ID Property Reference would indicate what characters are valid for an ID, whether the property is required, and other pertinent information related to IDs.

The following reference information can be used when designing a site:

- Changed_templates
- Contents_sync
- Applet_edit_box
- Edit_box_hooks
- Frames
- Frameset

- No_Options_Search_Form
- Saved_Search
- Set_Options_Search_Form
- Shipping_templates
- UserData_service
- UserInfo_component
- Word_wheel_applet_hooks
- Word_wheel_applet_query
- Word_wheel_applet

Changed Templates Reference

Some template files changed for NXT 3 version 3.4 from NXT 3 version 3.3. Most changes for NXT 3 version 3.4 were only to the color scheme for many templates and graphics.

The following table lists all templates that have changed, where the change occurs in the template file, and a brief description of the change.

Template File	Description
glossary.htm	File no longer ships with templates.
logo.pbnp.gif	File name changed from JPG extension to GIF entension.
ui-update-g.gif	New graphic for NXT 3 version 3.4. Used with Manage Content link in main template. Main template set previously used ui-update.gif. However, this previous graphic uses nearly the same color as the new color scheme. The ui-update.gif file is still used in other places where it does not occur on top of the new color scheme.

See Also

The complete list of shipping_templates

Contents Sync Reference

The Contents component allows you to select whether you want the table of contents to automatically sync with the current document.

sync

Possible values are 0, 1, or 2. The default is 0, which means not to automatically sync the table of contents and the document frame. 1 and 2 turn on automatic syncing. 1 syncs the table of contents. 2 syncs the table of contents and highlights the current node.

Edit Box Applet Reference

Methods

The edit box applet provides communication to a word wheel as the user types. The applet provides the following methods that can be called from JavaScript:

void SetFocusHook(String sFocusHook)
Sets the JavaScript function query for the element server. It will take effect when the next operation is performed.

void SetSelectHook(String sSelectHook)
Moves to the first element.

void SetSeekHook(String sSeekHook)
Moves to the last element.

void SetInsertHook(String sInsertHook)
Moves up a page.

void Insert(String sWord)
Moves down a page.

String GetText()
Gets the text from the edit box.

void SetText(String s)
Sets the text to the edit box.

Appearance

background
Background color of edit box; the default is white (0xf0f0f0).

textcolor
Text color of edit box; the default is black (0x000000).

fontname
Font name of edit box; the default is dialog.

fontsize
Font size of edit box; the default is 14.

Communication

Normally only one of the following parameters is set.

selecthook

The JavaScript function to call when an item is selected in the list. The default is ww_selecthook.

editbox
The name of edit box to originally be linked to. The default is EditBox.

See Also

Edit_Box_Hooks_Reference

Word_Wheel_Applet_Reference

Word_Wheel_Applet_Hooks_Reference

Word_Wheel_Applet_Query_Reference

Edit Box JavaScript Hooks

The edit box has the following hooks:

function eb_focushook() { document.WordWheel.SetQuery("*"); document.WordWheel.ToFirst(); }
This hook is called when the edit box receives focus. It is generally used to set the query in the word wheel.

function eb_inserthook(sTerm, sText, nTermStartIndex, nTermEndIndex, nSelectionStartIndex, nSelectionEndIndex) { return sText.substring(0, nTermStartIndex) + sTerm + sText.substring(nTermEndIndex + 1); }
This hook is called when an item needs to be inserted into the edit box. This can be in response to the user double clicking in the word wheel or pressing enter in the edit box. This function has access to the entire string in the edit box, the positions of the word the cursor is on, and the selection position. Using this information, it will construct the new string for the edit box and return it from this hook. The returned string will completely replace the contents of the edit box.

function eb_seekhook(sTerm, sText, nTermStartIndex, nTermEndIndex, nSelectionStartIndex, nSelectionEndIndex) { document.WordWheel.Seek(sTerm); }
This hook is called when the user moves the cursor, when the cursor moves to a different word, or the user edits word the cursor is on. Normally, this synchronizes the edit box and word wheel by setting the word wheel to the word the cursor is on. This hook has access to the entire string in the edit box, the position of the word the cursor is on, and the selection if it wishes to determine what word to seek to in a manner other than the default.

function eb_selecthook(sTerm, sText, nTermStartIndex, nTermEndIndex, nSelectionStartIndex, nSelectionEndIndex) { return document.WordWheel.GetCurrent(); }
This hook is called when the user presses the enter key in the edit box to indicate to insert the current item from the word wheel into the edit box. The string returned from this hook is inserted. Normally, the string is obtained by calling GetCurrent() on the word wheel. This function has access to the entire string

in the edit box and the positions of the word the cursor is on and the selection. Normally these would not be used in this function. If custom behavior is needed, it is usually best to do it in the inserthook.

See Also

Edit_Box_Applet_Reference

Word_Wheel_Applet_Reference

Word_Wheel_Applet_Hooks_Reference

Word_Wheel_Applet_Query_Reference

Frames Reference

Frame pages are defined using the FRAME element within the FRAMESET element. The most common parameters for the frame element are as follows:

name
Name of the frame page. Assigning the frame page a name provides a way for other frame pages to reference it.

src
NXT 3 URL to display in the frame page. The URL specifies the template file to display and variables (named values) to pass to the frame page. Page variables are typically used to pass a frame page the names and file names of other frame pages in the frame set. The frame page uses these variables to interact with the other frame pages.

frameborder
"Yes" or "no" value that determines whether or not the frame has visible borders. The default is "yes".

noresize
Specifies that users cannot resize the frame. Can be coded as `noresize` or `noresize="noresize"`. Default is to allow resize.

scrolling
Specifies whether scrollbars are available on a frame. Possible values are yes (scrollbars always available), no (never available), and auto (the Web browser determines if they are needed). Default is auto.

The following example shows a portion of the frameset's main page:

```
<frameset rows="80,*" border="0">
 <frame name="banner" src="banner.htm"
  scrolling="no" noresize="noresize">

 <frameset cols="200,*">
  <frame name="contents" src="contents.htm">
```

```
  <frame name="main" src="document.htm">
 </frameset>

<noframes>
<body>
<p>This site requires a frames-capable browser.</p>
</body>
</noframes>

</frameset>
```

Frameset Reference

A frameset's main page contains one or more FRAMESET elements defining the orientation and proportions of the **frame pages** it contains. You can use the NOFRAMES tag inside a FRAMESET tag to provide alternative content for browsers that cannot display frames.

> **Note:** An HTML document that contains a FRAMESET tag cannot contain a BODY tag (except inside the NOFRAMES element).

A frameset can specify that its frames are laid out in rows or columns. If you want your frameset to have rows and columns, rather than just rows or columns, you can use FRAMESET tags nested inside FRAMESET tags. For example, you could define a frameset that has two columns, where the first column contains a frameset that has two rows and the second column contains a frameset that has 4 rows.

When you define a link, (using the <A HREF> tag) you can specify in which frame the destination document is displayed, by giving the name of the frame as the value of the link's TARGET attribute.

The most common parameters for frameset are as follows:

cols
: A comma-separated list of values giving the width of each frame in the frameset. The values of each item can be entered in pixels, a percentage of the total, or an asterisk (*) that means to use as much space as possible from what remains.

rows
: A comma-separated list of values giving the height of each frame in the frameset. The values of each item can be entered in pixels, a percentage of the total, or an asterisk (*) that means to use as much space as possible from what remains.

border
: The thickness of frame borders for all frames in an outermost frameset. A setting of 0 causes all frames in the frameset to have no border between them. If no BORDER tag is present, the default is 5 pixels.

The following example shows a portion of a frame set's main page:

```
<frameset rows="80,*" border="0">
 <frame name="banner" src="frame1.htm">
 <frameset cols="200,*">
  <frame name="contents" src="frame2.htm">
  <frame name="main" src="frame3.htm">
 </frameset>

<noframes>
<body>
<p>This site requires a frames-capable browser.</p>
</body>
</noframes>

</frameset>
```

The result is a top frame that spans the width of the window and is 80 pixels tall and a bottom frame that takes up the rest of the window with a narrow (200 pixel) left frame and a wide right frame inside it.

last-modified Field Reference

All documents in the NXT 3 site automatically include the **last-modified** field.

This value indicates the last time that this document was changed, either through the Manage Content feature, or when the content collection was rebuilt.

You can use the values from this field in a search form, or in the search results list.

locked Field Reference

All documents in the NXT 3 site automatically include the **locked** field.

This value indicates whether a user has checked out the document through the Manage Content feature. Each document has a true or a false value for this field, indicating whether or not it is checked out.

You can use the values from this field in a search form, or in the search results list.

See Also

locked-by Field Reference

locked-by Field Reference

All documents in the NXT 3 site automatically include the **locked-by** field.

This value indicates the name of the user who has checked out the document through the Manage Content interface.

You can use the values from this field in a search form, or in the search results list.

See Also

locked_Field_Reference

No Options Search Form Reference

The simplest search form does not need any scripting. Users may enter their search criteria directly into an element named "xhitlist_q"; you (as the designer of the form) may set all other options within the form. Use the Set_Options_Search_Form if you want the user to be able to set options for the query.

This template sets all options internally. The user is allowed to enter the terms to be searched.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<HTML>
<HEAD>
<TITLE>Super Simple Search Form</TITLE>
</HEAD>

<BODY>

<FORM NAME="SuperSimpleSearch"
ACTION="<!-- #EXECUTIVE:SCRIPT_NAME -->"
METHOD="POST" TARGET="main">
<B><!-- Enter Search Here -->

<P><INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Search!">

<INPUT NAME="xhitlist_q" VALUE="" TYPE="TEXT" SIZE="20">
<INPUT NAME="f" VALUE="xhitlist" TYPE="HIDDEN" SIZE="0">
<INPUT NAME="xhitlist_d" VALUE="" TYPE="HIDDEN" SIZE="0">
<INPUT NAME="x" VALUE="Simple" TYPE="HIDDEN" SIZE="0">
<INPUT NAME="xhitlist_s" VALUE="relevance-weight"
TYPE="HIDDEN" SIZE="0">
<INPUT NAME="xhitlist_hc" VALUE="" TYPE="HIDDEN" SIZE="0">
<INPUT NAME="xhitlist_xsl" VALUE="xhitlist.xsl"
TYPE="HIDDEN" SIZE="0">
<INPUT NAME="xhitlist_vpc" VALUE="first"
TYPE="HIDDEN" SIZE="0">
<INPUT NAME="xhitlist_sel"
VALUE="title;path;relevance-weight;home-title"
TYPE="HIDDEN" SIZE="0">

</FORM>
```

```
</BODY>
</HTML>
```

Set Options Search Form Reference

This search form template not only allows the user to enter a search string, it also allows the user to set several search options. It uses a script to retrieve the specified values and assign them to the form's parameters. The search form shown here is the Advanced Search template.

```
<script type="text/javascript">

function countQueryTerms(term_list) {

if (term_list == "") {
 return 0;
}

term_count = 1;

last_index = 0;

while (last_index != -1 && last_index < term_list.length) {
   new_index = term_list.indexOf(" ", last_index);
    // count adjacent spaces only once
   if (new_index - last_index >= 1)
   {
    term_count++;
   }
    last_index = (new_index == -1)
  ? new_index : new_index + 1;
}
 return term_count;
 }
  function submitQuery() {
  var positive_terms = "";
  var negative_terms = "";
  var rank_terms = "";
   var ui = document.ui_form;
  var request = document.request_form;
   // 'and' terms
  if (ui.and.value != "")  {
   var and_input = ui.and.value;
   positive_terms += "[and:" + and_input + "]";
   rank_terms += and_input + " ";
  }
```

```
 // 'not' terms
if (ui.not.value != "")  {
 var not_input = ui.not.value;
 negative_terms += "[not:[or:" + not_input + "]]";
}
 // 'or' terms
if (ui.or.value != "")  {
 var or_input = ui.or.value;
 positive_terms += "[or:" + or_input + "]";
 rank_terms += or_input + " ";
}
 // phrase terms
if (ui.phrase.value != "")  {
 var phrase_input = ui.phrase.value;
 positive_terms += "[orderedprox,0:" + phrase_input + "]";
 rank_terms += "[orderedprox,0:" + phrase_input + "]";
}
 // proximity terms
if (ui.prox.value != "")  {
 var proximity_input = ui.prox.value;
    // The "proximity window" is the range that
  // determines whether  query terms are near
  // each other. We make this dynamic based on
  // the number of words the user is looking for.
 prox_window = countQueryTerms(ui.prox.value) * 10;
   positive_terms += "[windowprox," +
 prox_window + ':' + proximity_input + ']';
 rank_terms += "[windowprox," +
 prox_window + ':' + proximity_input + ']';
}
 // stemming
var stem_positive = "";
var stem_rank = "";
if (ui.stemming.checked && positive_terms != "")  {
 stem_positive = "[stem:" + positive_terms + "]";
 stem_rank = "[stem:" + rank_terms + "]";
}
  // thesaurus  var thesaurus_positive = "";
var thesaurus_rank = "";
if (ui.thesaurus.checked && positive_terms != "")  {
 thesaurus_positive =
  "[thesaurus:" + positive_terms + "]";
 thesaurus_rank = "[thesaurus:" + rank_terms + "]";
}
 // combine stemming/thesaurus
```

```
  if (stem_positive != "")  {
   positive_terms = stem_positive +
    ((thesaurus_positive != "") ?
    thesaurus_positive : "");
  }
  else if (thesaurus_positive != "")  {
   positive_terms = thesaurus_positive;
  }
   if (stem_rank != "")
  {
   rank_terms = stem_rank +
    ((thesaurus_rank != "") ? thesaurus_rank : "");
  }
  else if (thesaurus_rank != "")  {
   rank_terms = thesaurus_rank;
  }
   // put everything together
  var query = "";
  if (positive_terms != "")  {
   query = "[rank,100:" +
    "[domain:[and:" + positive_terms + negative_terms +
    "]][sum:" + rank_terms + ']]';
    request.xhitlist_s.value = "relevance-weight";
  }
  else if (negative_terms != "")  {
   query = negative_terms;
  }
   // set the variables in the other form (the one that will
 // be submitted)
  request.xhitlist_q.value = query;
   // kwic
  if (ui.kwic.options[ui.kwic.selectedIndex].value != 0)  {
   selected_index = ui.kwic.selectedIndex;
   var sorting = request.xhitlist_hc.value;
    request.xhitlist_hc.value = "[XML][kwic," +
     ui.kwic.options[selected_index].value +
     "]" +    sorting;
   request.xhitlist_sel.value += ";hit-context";
  }

<form
 name="request_form"
 action="#!-- #EXECUTIVE:SCRIPT_NAME --#"
 method="post"
 target="_self">
```

```
<input type="hidden" size="0" name="f" value="xhitlist">
<input type="hidden" size="0" name="xhitlist_q" value="">
<input type="hidden" size="0" name="xhitlist_x"
 value="Server">
<input type="hidden" size="0" name="xhitlist_s "value="">
<input type="hidden" size="0" name="xhitlist_hc" value="">
<input type="hidden" size="0" name="xhitlist_d" value="">
<input type="hidden" size="0" name="xhitlist_xsl"
 value="xhitlist.xsl">
<input type="hidden" size="0" name="xhitlist_vpc"
 value="first"/>
<input type="hidden" size="0" name="xhitlist_sel"
 value="title;path;content-type;home-title">
</form>
```

Shipping Templates Reference

NXT 3 includes several templates to help you get your site up and running quickly.

The following table identifies the files and templates provided with NXT 3 in the Templates directory. The table specifies the file name, the template or replacement variable calling the template, files called from within, and a brief description.

Name	Called By	Files Called	Description
advsearch.htm	#SEARCH-FORMS:ID	xhitlist.xsl	Advanced Search form
banner.css	banner.xsl		Formatting codes for banner.xsl.
banner.xsl	default.htm	banner.xsl contents-frame-h.htm contents-frame-j.htm doc-results.htm document-frameset.htm main.css searchform.htm sync-getstate.xsl update.xsl xhitlist.xsl	XSL used to create the main toolbar.

boolsearch.htm	#SEARCH-FORMS:ID	xhitlist.xsl	Boolean Search form.
challenge.htm	URL request	main.css	Authorization form. Prompts for user name and password.
contents-failed.htm		error-details.htm main.css	Error template. Displayed when the table of contents could not be populated.
contents-frame-h.htm	default.htm	document-frameset.htm main.css	HTML version of the table of contents.
contents-frame-j.htm	banner.xsl	document-frameset.htm	Java version of the table of contents. The document-frameset template is called when the user selects a table of contents entry.
contents-h.htm	#CONTENTS		Populates the HTML version of the table of contents with parent and children entries.
contents-invalidpath.htm		main.css	Error template. Displayed when the link to the entry selected is invalid.
contents-localmedia.htm		main.css	Error template. Displayed when an entry is selected that is designated as localmedia.
date-time.xsl	Stylesheets that need date and time formatting, statistics, manage content, and support		Generic date and time formatting stylesheet.

default.htm	URL request	banner.xsl contents-frame-h.htm document-frameset.htm tabs.js	Main frameset.
denied.htm			Error page. Displayed when the user tries to perform something for which they do not have rights.
doc-results.htm		document-frameset.htm xhitlist.xsl	Splits the document frame to display the document and search results windows. Used for the Doc/Results tab.
document.htm	Document replacement variable	main.css	Displays the selected document's content.
document-accessdenied.htm		error-details.htm main.css	Error page. Displayed when a user selects content for which they do not have rights.
document-accessdenied-concurrency.htm		error-details.htm main.css	Error page. Displayed when a user selects content for which the user limit has been reached.
document-accessdenied-expired.htm		error-details.htm main.css	Error page. Displayed when a user selects content for which the license has expired.

Document-failed.htm		error-details.htm main.css	Error page. Displayed when a document's content cannot be retrieved.
document-frame.htm	document-frameset.htm		Frame within the Document Frameset.
document-frameset.htm	default.htm	document-frame.htm document-tools.htm	Populates the document portion of the window with the document toolbar and document content.
document-localmedia.htm	id-localmedia.htm	contents-localmedia.htm main.css	Error page. Displayed when content specified as localmedia is not available. The user is prompted to insert the appropriate media and then choose Continue to access the content.
Document-Timeout.htm	#DOCUMENT	main.css	Error page. Displayed when process to load a document takes longer than the Document Timeout field specifies.
document-tools.htm	document-frameset.htm	document-frame.htm main.css reference.htm xhitlist.xsl	Places document tools in the document window. Tools include Sync Contents, Previous Match, Next Match, Clear Highlights, Find Similar, and Reference.

error-details.htm	contents-failed.htm document-accessdenied.htm document accessdenied-concurrency.htm document-accessdenied-expired.htm document-failed.htm reference-failed.htm		Displays the error value and message associated with the error received.
escape.htm		escape.js	Used by NextPage support. Not needed for site to run.
escape.js	Many templates		File containing scripts for UTF-8 encoding and decoding and URL escaping.
hit-tracker.js	document-tools.htm		File containing scripts for navigating hits within a document
id-localmedia.htm	#ID		Used to resolve IDs on content that is specified as localmedia when it is not available.

main.css	banner.xsl challenge.htm contents-failed.htm contents-frame-h.htm contents-invalidpath.htm contents-localmedia.htm document.htm document-accessdenied.htm document-accessdenied-concurrency.htm document-accessdenied-expired.htm document-failed.htm document-localmedia.htm document-tools.htm message-log.xsl reference.htm reference-failed.htm saved-search.xsl searchform.htm statistics.xsl titlepage.htm xhitlist.xsl		Formatting codes for templates.
message-log.xsl		main.css statistics.css	Displays Errors and Warnings in the statistics document.

metadatasearch.htm			Search form for document properties that match those in manage content.
quickfactssearchex-ample.htm	#SEARCH-FORMS:ID	quickfactssearchex-ample.xsl	Sample search form showing how to include a Java word wheel and limit searches to a specific domain.
quickfactssearchex-ample.xsl	quickfactssearchex-ample.htm	main.css tri-state-check.js escape.js	File for displaying the search results list from a Quick Facts search.
redirect-querylink.htm	QueryLink component	xhitlist.xsl	Redirects a query link within a migrated content collection to the Search Results frame.
reference.htm	document-tools.htm	main.css	Launches a separate window that displays the path to the current document.
reference-failed.htm		error-details.htm main.css	Error page. Used when reference information cannot be displayed for the current document.
saved-search.xsl	xhitlist.xsl	main.css	Displays the Manage Saved Searches window.
searchform.htm	banner.xsl	document-frameset.htm main.css	Lists the search forms available for the site.
statistics.css	message-log.xsl		Formatting codes for statistics document.

statistics.xsl		main.css message-log.xsl statistics.css	Template for Statistics page.
support.css			Used by NextPage support.
support.htm			Used by NextPage support.
support-bin.xsl			Used by NextPage support.
support-prop.xsl			Used by NextPage support.
support-sys.xsl			Used by NextPage support.
sync.css	sync-std.xsl		Formatting codes for sync-std.xsl.
sync-apply.xsl	sync-noop.xsl	sync-setresult.xsl sync-strings.xml sync-std.xsl	Used for synchronization process. Must not be modified.
sync-avail.xsl	sync-getstate.xsl	sync-strings.xml sync-std.xsl sync-date-time.xsl sync-get.xsl	Used for synchronization process. Must not be modified.
sync-date-time.xsl	sync-avail.xsl sync-get.xsl		Template used for displaying the date and time.
sync-get.xsl	sync-avail.xsl	sync-strings.xml sync-std.xsl sync-date-time.xsl	Error template. Used for synchronization process. Must not be modified.

sync-getstate.xsl	sync-setresult.xsl	sync-avail.xsl sync-strings.xml sync-std.xsl	Used for synchronization process. Must not be modified.
sync-noop.xsl	SYNC component	sync-apply.xsl sync-strings.xml sync-std.xsl	Used for synchronization process. Must not be modified.
sync-setresult.xsl	sync-apply.xsl	sync-getstate.xsl sync-strings.xml sync-std.xsl	Used for synchronization process. Must not be modified.
sync-std.xsl	sync-apply.xsl sync-avail.xsl sync-get.xsl sync-getstate.xsl sync-noop.xsl sync-setresult.xsl	main.css sync.css escape.js	Standard template for the sync interface.
sync-strings.xml	#EXECUTIVE sync-apply.xsl sync-avail.xsl sync-get.xsl sync-getstate.xsl sync-noop.xsl sync-setresult.xsl		File containing the standard sync messages.
tabs.js	default.htm		Generates tabs for default view.
titlepage.htm		main.css	Displays in the document window until content is selected.

tri-state-check.js	xhitlist.xsl		Creates the check boxes for specifying similar and dissimilar in the search results list.
update.css	update.xsl		Formatting codes for the update window.
update.js	update.xsl	update-help.js	Javascript code for Manage Content.
update.xsl	banner.xsl	update.css update.js update-adddoc-1.xsl update-checkin.xsl update-download.xsl update-props.xsl update-view.htm	Manage Content template.
update-adddoc-1.xsl	update.xsl	update.css update.js update-adddoc-2.xsl	Document for step 1 of the Manage Content Add Document process.
update-adddoc-2.xsl	update-adddoc-1.xsl	update.css update.js update-adddoc-1.xsl	Document for step 2 of the Manage Content Add Document process.
update-checkin.xsl	update.xsl	update.css update.js	Document displayed when checking in a document with Manage Content.
update-download.xsl	update.xsl	update.css update.js	Document displayed when checking out a document with Manage Content.

update-help.htm	update.js	update.css	Displays help information for Adding a Document, Document Check-in, and Editing Document Properties.
update-props.xsl	update.xsl	update.css update.js	Template for adding or editing properties for documents in Manage Content.
update-shortprops.xsl	update-view.htm	update.css update.js	Creates the document property portion of the Manage Content window when you view a document.
update-view.htm	update.xsl	document-frame.htm update-shortprops.xsl	Template for viewing documents within Manage Content.
user-limit.htm	#EXECUTIVE		Error page. Used when the maximum number of users based on the server license has been exceeded.
xhitlist.xsl	Search forms	banner.xsl document-frameset.htm main.css saved-search.xsl tri-state-check.js	Search Result List template.

Table of Contents Applet Reference

The Java Applet for the table of contents has two separate parts, the applet code for inserting the applet in the page and the code for the optional parameters.

Applet Code

The code for inserting the applet in the page should be entered exactly as follows (the width and height values can be changed as needed):

```
<applet
  code="JTOC.class"
  archive="JTOC.jar"
  codebase="/NXTApplets"
  name="JTOC"
  width="100%"
  height="99%"
  mayscript="mayscript">
```

Note: Using height="100%" in the applet tag causes the window's scrollbar to appear in Netscape, so you end up with a double scrollbar - one from the window, and one from the applet.

Applet Parameters

Refer to **Java Table of Contents Parameters** in *Requesting Content from NXT 3 Help* for information about the possible parameters and their values.

The resulting HTML code could look like the following for a Java table of contents:

```
<applet
  code="JTOC.class"
  archive="JTOC.jar"
  codebase="/NXTApplets"
  name="JTOC"
  width="100%"
  height="99%"
  mayscript="mayscript">

  <param name="Target" value="main">
  <param name="ExtDll" value="#!-- #EXECUTIVE:SCRIPT_NAME --#">
  <param name="DocTemplate" value="document-frameset.htm">
  <param name="SyncPath" value="">
  <param name="TextColor" value="#000000">
  <param name="LineColor" value="#666666">
  <param name="VisitedTextColor" value="#006666">
  <param name="BackgroundColor" value="#cccccc">
  <param name="MouseOverColor" value="#000099">
  <param name="SelectedBackColor" value="#000099">
  <param name="SelectedTextColor" value="#cccccc">
  <param name="ShowImages" value="1">
  <param name="ShowCheckBoxes" value="0">
  <param name="OnClickHook" value="onClick">
  <param name="LangLoadingNode" value="Loading...">
  <param name="LangMoreNode" value="More...">
  <param name="LangProcessingData" value="Processing Data...">
```

```
<param name="LangErrorNoServerData" value="No data returned from server."
<param name="LangLoadedNDocuments" value="Loaded %i documents.">
<param name="LangSynchronizingTOC" value="Synchronizing the TOC...">
<param name="LangRequestingChildren" value="Requesting children...">
<param name="LangAddingDocument" value="Adding document %s">
<param name="LangErrorNoTitle" value="Error - Child with no name or title
<param name="LangServerError" value="Error returned from server.">
```

UserData Service Reference

The UserData Service manages user settings for other services and components. It exposes a properties object for the current user, giving these services and components access to a set of name-value pairs for each user. The current user is determined by a call to access control. If access control is not implemented, the cookie ID is used instead of the user name. Cookies do have the drawback of not being accessible from another machine or multiple browsers, so if users want to access the site from multiple machines or browsers, access control provides the ability to store the user's information and does not require a cookie.

The UserData Service stores user setting through the DataStore Service listed in the UserData.ini file. The following shows the UserData.ini file (found in the bin directory) and explains what needs to be included for the data store to be recognized.

```
[] service=  [service] instance= [service|instance]
userdata=  [Service|Instance|UserData]
DatabaseService=<DataStoreName>
DatabaseInstance=<DBInstance>
```

DataStoreName
: Name of a service listed in the executive.ini file that implements data stores. For example, FileData.

DBInstance
: Name of the instance of the DataStoreName that you want to use. The instance name must appear in the <DataStoreName>.ini file. For example, if the File-Data.ini has a Service Instance defined as UserData, then you would replace <DBInstance> with `UserData`.

The UserData Service uses the FileData Service to store the users settings. The file used to store the user data is designated in the <DataStoreName>.ini file. For example, the shipping NXT 3 product includes FileData.ini file for the FileData Service. This FileData.ini includes a reference to the userdata.fda file. Keep in mind that the FileData.ini file contains additional service instances.

UserInfo Component Reference

The UserInfo component allows user preferences to be stored as an XML document. This XML document is created and modified using the UserInfo URL interface. The XML

document is passed to the UserData Service for storage. The UserInfo component receives a request from the Executive to process user data. The request from the Executive requires an initial interaction with the UserData service to obtain the user property object. If Access Control is not activated or the user's name cannot be determined, the information, such as user name, is pulled from the cookie ID.

The following list identifies the URL requests handled by the UserInfo component.

f
Function used to route the URL to the UserInfo component.

UserInfo_cat
Top level XML tag.

UserInfo_c
Command directive for the UserInfo component. Values are:
add: adds an item. Requires the **n** and **ipl** parameters. If the item already exists, the existing item is overwritten with the new data.
delete: deletes an item. Requires the **n** parameter.
rename: renames an item. Requires the **n** and **nn** parameters.
nuke: removes the XML associated with the **cat** parameter.

UserInfo_n
Name of the item to add, delete, or rename.

UserInfo_ipl
Item Property List. Semicolon delimited list of item properties to add. If a URL contains one of the item property names as a parameter, the parameter's value becomes the item property's value.

UserInfo_ItemPropertyName
Specifies an item property's value for an add command.

UserInfo_nn
New Name. Used with the rename command to specify the item's new name.

UserInfo_redirect
Specifies the URL to redirect to.

UserInfo_xsl
Stylesheet file name. Specifies the name of the stylesheet to use when processing the XML. This parameter can be used with or without the **c** parameter. The style sheet must be located in the **Style Sheet** directory specified in the Content Network Manager. If the redirect parameter is included, this parameter is meaningless.

The previous list identified how and what is stored as user preferences by the UserInfo component. Conceptually, the storage of user data is organized hierarchically. A template or style sheet developer determines the category or categories that will be available to the user. Each category is made up of items such as saved searches. Within an item are attributes. The attributes are the specific data needed to recreate the

saved user data. For example, with saved searches, the attributes could be the domain, the query type (advanced or simple), syntax, and so forth. Each category can contain multiple items and each item can contain multiple attributes. A user may have access to one or more categories. This defines the structure of the actual XML.

As a template or stylesheet developer, keep in mind that the storing of content must conform to this type of hierarchy for the XML to be processed correctly. The XSL you create determines how the XML is displayed and the category name.

Java Word Wheel JavaScript Hooks

The word wheel applet has one JavaScript hook:

function ww_selecthook(sTerm) { document.EditBox.Insert(sTerm); }
: This hook is called when the user double clicks in the word wheel indicating that a term is to be inserted into the edit box. The current term in the word wheel is passed to the function. Normally, the term is inserted into the edit box by calling the insert function. The insert function will in turn call the inserthook if it has been set, or insert the word in the default way if it hasn't, replacing the word the cursor is on with the new word.

See Also

Edit_Box_Applet_Reference

Edit_Box_Hooks_Reference

Word_Wheel_Applet_Reference

Word_Wheel_Applet_Query_Reference

Java Word Wheel Applet Query Reference

The query has one of the following forms:

*****
: Lists all terms in the site.

a*
: Lists all terms in the site that begin with the letter a.

[Field,*]
: Lists all fields in the site.

[Field,a*]
: Lists all fields in the site that begin with the letter a.

[Field,Author:*]
: Lists all the contents of the author field.

[Field,Author:a*]
: Lists all the contents of the author field that begin with the letter a.

See Also

Edit_Box_Applet_Reference

Edit_Box_Hooks_Reference

Word_Wheel_Applet_Reference

Word_Wheel_Applet_Hooks_Reference

Java Word Wheel Applet Reference

The word wheel applet provides a list of elements from the NXT 3 Server. The elements could be field names, field values, or words. The applet supports large lists by loading the lists in sections. The applet provides the following methods that can be called from JavaScript:

Methods

SetQuery(String sQuery)
Sets the query for the element server. The set takes effect when the next operation is performed.

void ToFirst()
Moves to the first element.

void ToLast()
Moves to the last element.

void PageUp()
Moves up a page.

void PageDown()
Moves down a page.

void LineUp()
Moves up a line.

void LineDown()
Moves down a line.

void Seek(String sWord)
Seeks to a certain word.

String GetCurrent()
Returns the currently selected item.

void SetDomain(String strDomain)
Sets the domain for items to be retrieved from.

void SetSelectHook(String sSelectHook)
Set the JavaScript function to call when an item is selected.

Appearance

background
Default is white (0xf0f0f0)

buttoncolor
Default is dark gray (0xb0b0b0)

scrollcolor
Default is light gray (0xd0d0d0)

bordercolor
Default is black (0xb0b0b0)

textcolor
Default is black (0x000000)

selectcolor
Default is light gray (0xe0e0e0)

selecttextcolor
Default is black (0x000000)

fontname
Default is dialog

fontsize
Default is 14

Operation

cachesize
The maximum number of list items to cache. The default is to cache all items.

chunksize
The number of items to load at a time. Increasing this value will cause requests to be made to the server less often, but the requests will take longer. The default value is 200.

Domain
The domain to limit to when getting items. It must be specified in NXT 3 domain syntax.

extdll
Default is `#!-- #EXECUTIVE:SCRIPT_NAME --#`.

query
The query to determine what items to display. It must be specified in NXT 3 Server query syntax.

Selectable
`1` keeps track of a highlighted item (default), `0` means no highlighted item.

Debug
`1` turns debug mode on (default), `0` turns it off.

Communication

Normally you use either the hooks or the wordwheel and query parameters, but not both the hooks and the parameters.

focushook
The JavaScript function to call when the edit box receives focus. The default is "eb_focushook".

selecthook
The JavaScript function to call when a word in the edit box is selected. The default is "eb_selecthook".

seekhook
The JavaScript function to call when the word the cursor is on has changed. The default is "eb_seekhook".

inserthook
The JavaScript function to call when the word is to be inserted in the edit box. The default is "eb_inserthook".

wordwheel
The wordwheel to associate with this edit box. The default is "WordWheel".

query
The query to send to word wheel. The default is "*".

See Also

Edit_Box_Applet_Reference

Edit_Box_Hooks_Reference

Word_Wheel_Applet_Hooks_Reference

Word_Wheel_Applet_Query_Reference

XSL Transformations Reference

NXT 3 version 3.4 uses the Microsoft XML Parser Version 3.0 (MSXML3) to perform server-side XSL transformations.

The NXT 3 site templates conform to the W3C XSL Transformations Version 1.0 recommendation, and not Microsoft's working draft implementation. Since MSXML3 supports both XSL versions, it is possible for you to use Microsoft working draft XSL stylesheets on an NXT 3 site, but NextPage recommends following the W3C recommendation instead. It is possible that future versions of NXT may use a different XSL processor. Newer versions of Microsoft XSL processors are likely to discontinue support for the older working draft implementation, and there are no non-Microsoft processors that support it.

Site Design Examples

Examples provide specific examples on how to use a feature.

- Creating multiple sites example
- Creating a multi-field search form example
- Using the Java Word Wheel example
- Saving a search example

Creating Multiple Sites Example

Consider the following example of a simple corporate site that consists of two content collections. Each of these content collections contain both sensitive and non-sensitive documents. For this example, assume that the names and the titles are the same.

In order to host this data and maintain the proper security for each group of users, four different sites must be created:

- Internal Company Site (internal)
- HR Employees Site (internal)
- Executive Committee Site (internal)
- General Public Site (public)

In this example, we will assume that the three internal sites all have the same interface. The Public site will look different.

The first step is to define the views that you want to be accessible from each site and the associated domain. Use the Content Network Manager application to do this. The Content Network Manager can be used to configure local and remote NXT 3 sites.

See Restrict access to specific content in *Content Network Manager Help* for instructions on how to specify a domain for the view.

To create the four basic site layouts described in this example, we define four views: one for each site. The domain specifies the content within the site that each view makes available. For example, if you want the HRData view to only display the Employee Directory and Job Openings documents, select these two content collections in the domain. When a user goes to the HRData view, they will only see the Employee Directory and Job Openings documents. If nothing is specified for the domain, all documents in the view are visible.

Once the site definition file is saved with the view information and the domain settings, users accessing each site will only see the items included in the domain specified in that site's view.

NXT 3 uses templates to define the interface of the sites it hosts. Each site has a set of default templates. In addition, each view may use different templates. Use a view element's templates attribute to specify the directory containing the templates to use for the view.

NXT 3 determines the templates, images, and stylesheets directories to use to retrieve a template using a combination of the directories specified on the content collection or folder node, the directories specified on the site node, and the directories specified on the view node. The various ways in which these are combined depend on whether the paths specified are relative or full, and whether a site-wide or content related file is being retrieved. A site-wide file is one that is used by the entire site or view, such as a template for the search results list, contents, or toolbar frames, or an image that is displayed in one of those frames. These files are the same for the entire view regardless of what content is being browsed in the document control. A content related file is one that is

specific to the content being viewed, such as the document template or a style sheet for a specific content collection.

For a site-wide file, the file is found in the following way:

If the view directory is absolute, it is used as is. If the view directory is relative, it is appended to the directory specified on the site node to make a full path.

For a content-specific file, the file is found in the following way:

For the content collection element and each of its parents in the site, the appropriate path property (i.e. templates, images, etc.) is read. If the property exists on any parent then the path will be checked. If the view directory is relative, it is appended to this path. Otherwise the path is used as is. For each parent, if the file exists in the path that file is used, otherwise it proceeds to the next parent, ultimately stopping at the directories specified on the site node.

The following examples show the four combinations possible for locating templates in a site. These examples are based on the following site definition file settings:

NXT 3 (Default Site) Node Properties

```
Templates directory=
c:\Program Files\NextPage\NXT 3\Templates
```

English view Properties

```
Templates directory=Enu
```

German view Properties

```
Templates directory=
c:\Program Files\NextPage\NXT 3\Templates\Deu
```

Example 1: Site-wide setting with relative path

In this example, a request is made for the English document-frame.htm file. The English view's relative path is appended to the site node's path resulting in the following:

```
c:\Program Files\NextPage\NXT 3\Templates\Enu\document-
frame.htm
```

Example 2: Site-wide setting with absolute path

In this example, a request is made for the German document-frame.htm file. Since the German view specifies an absolute path, the following path is used for the request:

```
c:\Program Files\NextPage\NXT 3\Templates\Deu\document-
frame.htm
```

Example 3: Content specific setting with relative path

In this example, a request is made for the English document-frame.htm file. The English view's relative path is appended to the site node's path resulting in the following:

```
c:\Program Files\NextPage\NXT 3\Templates\Enu\document-
frame.htm
```

This directory is checked first for the document-frame.htm file. If not found, it moves to the parent directory, which is c:\Program Files\NextPage\NXT 3\Templates. The c:\Program Files\NextPage\NXT 3\Templates directory is the root directory of the site, so no further checking is performed.

Example 4: Content specific setting with absolute path

In this example, a request is made for the German document-frame.htm file. Since the German view specifies an absolute path, the site node's path is used to make the following request:

```
c:\Program Files\NextPage\NXT 3\Templates\document-frame.htm
```

Creating a Multi-Field Search Form Example

This example shows a multi-field search form, as well as a script to handle the search string processing. It does not allow the user to set any options. (See the Set Options Search Form task for instructions about setting search options from the template.)

```
<SCRIPT LANGUAGE="javascript">
<!-- Code for unemployment and inflation field search -->

function submitSearch() {
 // build the query string
  if (document.UnemploymentSearchForm.greaterless[0].checked == "1")  {
  document.UnemploymentSearchForm.xhitlist_q.value="[Field " +
   document.UnemploymentSearchForm.fieldname.options[
   document.UnemploymentSearchForm.fieldname.selectedIndex].value +
   ": >" +
   document.UnemploymentSearchForm.rate.value + "]";
 }
  else {
  document.UnemploymentSearchForm.xhitlist_q.value="[Field " +
   document.UnemploymentSearchForm.fieldname.options[
   document.UnemploymentSearchForm.fieldname.selectedIndex].value
+
   ": <" +
   document.UnemploymentSearchForm.rate.value + "]";
 }
  // if they are searching on unemployment instead of inflation
  if (document.UnemploymentSearchForm.fieldname.value == "unemployment") {
  // Change sort order
  document.UnemploymentSearchForm.xhitlist_s.value="Field:Unemployment,Desc
  document.UnemploymentSearchForm.xhitlist_sel.value=
   "title;path;field:unemployment;field:inflation";
 }
}
<!-- end of code for unemployment and inflation search -->
</SCRIPT>

<!-- Form for searching the unemployment and inflation fields -->

<h1>Country Quick Facts</h1>
<h2>Unemployment and Inflation Search</h2>
<table border="0" cellPadding=3 cellSpacing="0">
<form name="UnemploymentSearchForm" onsubmit="submitSearch()"
 action="#!-- #EXECUTIVE:SCRIPT_NAME --#?f=xhitlist"
 method="post" target="main">
<!-- This is the part of the form that sends the values the server
```

```
cares about for executing the query. -->
<input type="hidden" size="0" name="f" value="xhitlist" />
<input type="hidden" size="0" name="xhitlist_pcd" value="" />
<input type="hidden" size="0" name="xhitlist_ncd" value="" />
<input type="hidden" size="0" name="xhitlist_vpc" value="first" />
<input type="hidden" size="0" name="xhitlist_xsl"
 value="QuickFactsSearchExample.xsl" />
<input type="hidden" size="0" name="xhitlist_d"
 value="{10.1048/NXT3/3.0/QuickFacts}" />
<input type="hidden" size="0" name="xhitlist_s"
 value="Field:Inflation,Desc"/>
<input type="hidden" size="0" name="xhitlist_x" value="Advanced" />
<input type="hidden" size="0" name="xhitlist_hc" value="" />
<input type="hidden" size="0" name="xhitlist_q" value="" />
<input type="hidden" size="0" name="xhitlist_sel"
 value="title;path;field:inflation;field:unemployment"/>

<tbody>

<!-- This part of the form provides the search user interface.
  The user's inputs are extracted from this form to create
 the query that will be sent to the server. -->

 <tr>
  <td></td>
  </tr>
 <tr>
  <td><font face=Arial>
   Search for countries with an 
   <select name="fieldname" size="2" class="control">
    <option selected value="inflation">inflation</option>
    <option value="unemployment">unemployment</option>
   </select>
    rate<br><br>
   <input type="radio" name="greaterless" value="greater"
    checked>greater than
   <input type="radio" name="greaterless" value="less"> less than
   <input size="30" name="rate" value="" />

   percent</font><br>
  </td>

</tr>
<tr>
  <td>
```

```
  <input type="submit" value="Search" class="button">  
  <input type="button" value="Cancel"
   onClick="cancelQuery('#!-- #EXECUTIVE:PARAM:G_MAIN_FRAMESET --#')"
   class="button">
  <td>

</tr>
 </tbody>
</form>
</table>
<!-- end of unemployment and inflation field search -->
```

Using the Java Word Wheel Example

The following example (JavaWordWheel.htm) shows a Java word wheel that lists the terms for the **Resource** field found in the Country Quick Facts content collection (QuickFacts.nfo).

```
<HTML>
<HEAD>
<TITLE>NXT 3 Java Word Wheel Search Form</TITLE>
<link rel="stylesheet" type="text/css" href="#!-- #STYLESHEETS:main.css --#
<SCRIPT type="text/javascript">

<!-- Code for searches using the Resource field -->

function ResourceJWWHook(sTerm) {
 document.ResourceEditBox.Insert(sTerm + ", ");
}

function ResourceJEBHook(sTerm) {
 document.ResourceWordWheel.Seek(sTerm);
}

function ResourceSearchForm_onsubmit() {
 var rquery;
  var rsrcterm;
  var i = 0;
  var rtext = document.ResourceEditBox.getText();

 var len = rtext.length;
  var start;

  var end;
   rquery = "[Field RESOURCE: ";

 while (i < len) {
   while ((i < len) && ((rtext.charAt(i) == ',') ||
   (document.ResourceSearchForm.xhitlist_q.value.charAt(i) == ' '))) {
   i++;
  }
   start = i;
   i++;
   while ((i < len) && (rtext.charAt(i) != ',')) {
    i++;
   }
  end = i - 1;
  if (start != end) {
```

```
    rsrcterm = document.ResourceEditBox.getText().substring(start,end+1);
   rquery += "\"" + rsrcterm + "\" ";
   if (rtext.charAt(i) == ',') {
    if (document.ResourceSearchForm.SearchType[0].checked) {
      rquery += "| ";
     }
     else {
     rquery += "& ";
      }
    }
   }
  }
 rquery += "]";
 document.ResourceSearchForm.xhitlist_q.value = rquery;
}
</SCRIPT>
</HEAD>

<BODY>

<!-- Form for Resource field search -->

<H1>Country Quick Facts</H1>
<H2>Natural Resource Search</H2>
<form name="ResourceSearchForm" onsubmit="ResourceSearchForm_onsubmit()"
 action="#!-- #EXECUTIVE:SCRIPT_NAME --#?f=xhitlist"
 method="post" target="main">
<span class="label">

<input type="hidden" size="0" name="f" value="xhitlist" />
<input type="hidden" size="0" name="xhitlist_pcd" value="" />
<input type="hidden" size="0" name="xhitlist_ncd" value="" />
<input type="hidden" size="0" name="xhitlist_vpc" value="first" />
<input type="hidden" size="0" name="xhitlist_xsl"
 value="QuickFactsSearchExample.xsl"/>
<input type="hidden" size="0" name="xhitlist_d"
 value="{10.1048/NXT3/3.2/QuickFacts}"/>
<input type="hidden" size="0" name="xhitlist_s" value="Relevance-Weight"/>
<input type="hidden" size="0" name="xhitlist_x" value="Advanced" />
<input type="hidden" size="0" name="xhitlist_hc" value="" />
<input type="hidden" size="0" name="xhitlist_q" value="" />
<input type="hidden" size="0" name="xhitlist_sel" value="title;path" />

<TABLE border=1 cellPadding=5 cellSpacing=1 cols=2>
<TR>
```

```
<TD>
<P>Search for the following Natural Resources:<BR>
(separate resources with a comma)
<APPLET
 archive="JWW.jar"
 code=EditBox.class
 codeBase="/NXTApplets"
 name="ResourceEditBox"
 width=300
 height=22
 MAYSCRIPT>
<!-- Appearance Parameters -->
<param name=background value="0xf0f0f0">
<param name=bordercolor value="0xb0b0b0">
<param name=textcolor value="0x000000">

<!-- Operation Parameters -->
<PARAM NAME=wordwheel VALUE="ResourceWordWheel">
<PARAM NAME=query VALUE="[Field, RESOURCE:*]">
</APPLET>
<P><INPUT name="SearchType" type=radio CHECKED>
 Find countries with <STRONG>ANY</STRONG> of these resources.<BR><BR>
<INPUT name="SearchType" type=radio>
 Find countries with <STRONG>ALL</STRONG> of these resources.</P>
<P align="center"><INPUT class=button type=submit value="Search Now"></P></


<TD width=200>
<P>Double-click a resource to add it to the search list:</P>
<P align="center">
 <APPLET
  archive="JWW.jar"
  code="WordWheel.class"
  codebase="/NXTApplets"
  name="ResourceWordWheel"
  width=150
   height=200
  MAYSCRIPT>
<!-- Appearance Parameters -->
  <param name=background value="0xf0f0f0">
  <param name=buttoncolor value="0xb0b0b0">
  <param name=scrollcolor value="0xd0d0d0">
  <param name=bordercolor value="0xb0b0b0">
  <param name=textcolor value="0x000000">
  <param name=selectcolor value="0xe0e0e0">
```

```
  <param name=selecttextcolor value="0x000000">

<!-- Operation Parameters -->
  <PARAM NAME=cachesize VALUE="500">
  <PARAM NAME=chunksize VALUE="20">
  <PARAM NAME=domain VALUE="{10.1048/NXT3/3.0/QuickFacts}">
  <PARAM NAME=hook VALUE="ResourceJWWHook">
  <PARAM NAME=extdll VALUE="#!-- #EXECUTIVE:SCRIPT_NAME --#">
  <PARAM NAME=query VALUE="[Field, RESOURCE: *]">
  <PARAM NAME=selectable VALUE="1">
  <param NAME="editbox "VALUE="ResourceEditBox">
</APPLET>
</P></TD>
</TR></TABLE></span>

</form>
```

Saving Search Queries Example

The following shows a portion of code included in the shipping templates. This code provides the interface for saved searches and managing saved searches, both of which interact with user data. The first section of code is the functions for saved searches and managing saved searches.

```
function saveSearch() {
 // clear the edit to make it look like something happened
 document.named_search_form.userinfo_n.value =
  document.named_search_form.title.value;
 document.named_search_form.title.value = "";
 return true;
}

function manageSavedSearches() {
 var new_window =
  window.open("<!-- #EXECUTIVE:SCRIPT_NAME -->
   ?f=userinfo$userinfo_cat=saved-search$userinfo_xsl=saved-search.xsl",
   "savedsearch",
   "width=500,height=500,resizable=yes,scrollbars=yes");
 new_window.focus();
}
```

This portion of the code creates a form that provides an edit box for the user to specify the name for the saved search. The code also shows the linked text for launching the Manage Saved Searches dialog box.

```
<!-- Saved Searches -->
 <form name="named_search_form" onsubmit="return saveSearch()"
  target="banner">
  <xsl:attribute name="action"><!-- #EXECUTIVE:SCRIPT_NAME --></xsl:attribu
   <table cellspacing="5" width="80%" nowrap="nowrap">
    <tr nowrap="nowrap">
     <td align="right" nowrap="nowrap">Save search as:</td>
     <td nowrap="nowrap">
      <input type="hidden" name="f" value="userinfo"/>
      <input type="hidden" name="userinfo_cat "value="saved-search"/>
      <input type="hidden" name="userinfo_c" value="add"/>
      <input type="hidden" name="userinfo_n" value=""/>
      <input type="hidden" name="userinfo_xsl" value="banner.xsl"/>
      <input type="hidden" name="userinfo_ipl"
       value="query;syntax;domain;sort;hit-context;select;stylesheet"/>
      <input type="hidden" name="userinfo_stylesheet" value="xhitlist.xsl"/
      <input type="hidden" name="userinfo_query">
       <xsl:attribute name="value">
        <xsl:value-of select="list-section/query"/>
```

```
      </xsl:attribute>
     </input>
     <input type="hidden" name="userinfo_syntax">
      <xsl:attribute name="value">
       <xsl:value-of select="list-section/query-syntax"/>
      </xsl:attribute>
     </input>
     <input type="hidden" name="userinfo_domain">
      <xsl:attribute name="value">
       <xsl:value-of select="list-section/domain"/>
      </xsl:attribute>
     </input>
     <input type="hidden" name="userinfo_sort">
      <xsl:attribute name="value">
       <xsl:value-of select="list-section/sort"/>
      </xsl:attribute>
     </input>
     <input type="hidden" name="userinfo_hit-context">
      <xsl:attribute name="value">
       <xsl:value-of select="list-section/hit-context-param"/>
      </xsl:attribute>
     </input>
     <input type="hidden" name="userinfo_select">
      <xsl:attribute name="value">
       <xsl:value-of select="list-section/select"/>
      </xsl:attribute>
     </input>

     <input type="text" size="20" name="title" maxlength="25"/>
     <input class="button" type="submit" value="save"/>
    </td>
    <td nowrap="nowrap">
     <a href="javascript:manageSavedSearches()" class="button">
      <img border="0" align="top">
       <xsl:attribute name="src">
        <!-- #IMAGES:ui-saved-search.gif -->
       </xsl:attribute>
      </img>Manage Saved Searches
     </a>
    </td>
   </tr>
  </table>
 </form>
```

The first set of INPUT tags uses the **UserInfo** component parameters to save the information stored in the form's edit box. Additional INPUT tags specify the specific

attribute to assign data to within the user data file. For example, the following code specifies syntax as an attribute within **UserInfo**.

```
<input type="hidden" name="userinfo_syntax">
<xsl:attribute name="value">
< xsl:value-of select="list-section/query-syntax"/>
</xsl:attribute>
```

The XSL tags translate the data from the edit box into the stored XML format.

The following shows the XML output generated by the code found in banners.xsl.

```
<?xml version="1.0"?>
<?xml-stylesheet
 type="text/xsl"
 href="/NXT/gateway.dll?f=stylesheets$fn=banner.xsl$3.0"?>
<saved-search>
 <item name="test">
  <query>the</query>
  <syntax>Simple</syntax>
  <domain></domain>
  <sort>relevance-weight</sort>
  <hit-context></hit-context>
  <select>title;path;relevance-weight;content-type;home-title</select>
  <stylesheet>xhitlist.xsl</stylesheet>
 </item>
</saved-search>
```

The **UserInfo component** and **UserData Service** provide flexibility and additional functionality for users of NXT 3.

If the user's browser does not support XML, the transformation is done server-side, and HTML is returned instead of XML. Notice also that the XML header is not saved from request to request. Each subsequent request modifies the user's XML document.