Table of Contents

Table of Contents .............................................................. 3
Introduction ............................................................................. 5
Portal Components .............................................................. 6
Creating Module Types ......................................................... 8
Manifest Definition ............................................................. 12
Creating Modules ............................................................... 17
Introduction

Overview

Blackboard Community System™ includes the ability to define Portal components through Building Blocks. A Building Block is a small collection of JavaServer Pages (JSP) that third party developers create to extend the functionality of the core Blackboard platform.

This document will help developers create modules for the Blackboard Community System that are secure, consistent with the Blackboard Academic Suite look and feel, integrate with the core application, and operate with external systems.

Audience

This document is intended for developers who are building or planning to build modules and module types to be added to the Blackboard Community System. Familiarity with both Java® and JavaServer Pages (JSP) is assumed. Additionally, working knowledge of the Blackboard Building Blocks APIs (Application Program Interfaces) and Building Block framework is helpful. For more information on these please refer to the Introduction to the Building Blocks APIs and Runtime and the Building Block Developer Guide.

Manual Updates

Please note that the Blackboard Building Blocks Module Developer Guide is updated periodically. Check the Date Last Update at the beginning of the manual to ensure that it is the most recent copy. Any updates are listed in the Appendix.

To report any comments or suggestions regarding this manual, please contact Blackboard Support.
Portal Components

Introduction

The Portal is comprised of several elements visible to the end user. The image below displays the different elements in the portal.

Tab

Module

Module Edit

Modules

A module is a discrete data object that is associated with a module tab area in the Blackboard Academic Suite. It is the most obvious organizing element on the tab area. Customized modules and module tab areas (other than the My Institution tab area) are only available with the Blackboard Community System.
Module Types

A Module Type is the code required to display data for modules. Every module has an associated Type that defines the code for viewing and managing the individual data modules. Typically, a Module Type will exist as one or more JSPs deployed in a Web application. Module Types are comprised of the following types of pages:

**View Page**
The View page is simply a servlet resource that is called in-line when the module for a given type is rendered. Since the page is called in-line, no HTML header or body tags are allowed. The output is displayed in the context of a module (as seen in the figure above).

**Admin Page**
Admin pages are the components used to edit module data that is globally available.

**Edit Page**
Edit pages are the components used to customize module data for specific users.

Channels

A Channel is a separate piece of data used by one of the Blackboard Community System-provided module types, RSS-Channel. RSS Channel modules may reference Channel definitions.

Tabs

Tabs are the organizers of the portal framework, providing top-level navigation points for the user. The tab frame remains constant throughout the user experience.

**Note:** While tabs are not directly exposed to third party module developers, they are important to the overall framework. Administrators may create tabs that point directly to a tool installed by a Building Block.
Creating Module Types

Overview
Module Types are pieces of JSP/Java logic that can display custom data on the portal page. Module Types include View, Edit, and Admin pages. These pages can contain Java code that access the Portal data API and any of the other public Blackboard APIs.

Implementing the JSPs
Technically the View, Edit, and Admin components can be any Web application resource. However, JSPs are the most useful because there are custom tags that wrap the navigation components.

Tags
Four tags are provided for easing the development of Edit and Admin pages. They are included in the standard Tag Library Descriptor (TLD). Other Blackboard tags may be used in conjunction with these tags.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modulePersonalizationPage</td>
<td>Renders the common page elements, including document template, navigation path, and title for module personalization.</td>
</tr>
<tr>
<td>modulePersonalizationReceipt</td>
<td>Common receipt after processing user personalization. It generates appropriate navigation elements.</td>
</tr>
<tr>
<td>moduleAdminPage</td>
<td>Renders the common page elements, including document template, navigation path, and title for editing a module.</td>
</tr>
<tr>
<td>moduleAdminReceipt</td>
<td>Common receipt after processing administrative editing. It generates appropriate navigation elements.</td>
</tr>
</tbody>
</table>
The following example shows how to use tags included in the TLD.

```xml
<bbUI:modulePersonalizationPage>
  <FORM method="post" action="proc_edit.jsp">
    <bbUI:step title="Edit Zip Code" number="1">
      <bbUI:dataElement label="Zip Code">
        <input TYPE="TEXT" NAME="zip">
      </bbUI:dataElement>
    </bbUI:step>
    <bbUI:stepSubmit title="Submit" number="2" />
  </FORM>
</bbUI:modulePersonalizationPage>
```

### Portal API

An API is provided to access Module and Personalization data from within Module Types.

The Portal API provides the capability to store blocks of data in the database. The principal object is the `CustomData` object.

**Note:** CustomData is stored in the database for each user and each module that invokes it. CustomData information is only removed when either the module or the user associated with the CustomData information is deleted from the system. Be aware that, in unusual circumstances, this could result in some larger than anticipated database growth.

### Custom Data

The `CustomData` object is used to get both module-specific and personalization data. The class is in the `blackboard.portal.external` package. The data may contain arbitrary name/value pairs that are retrieved via `getValue()` and `setValue()` methods.

A static method is provided for obtaining an instance of the `CustomData` object initialized for the module being processed, `CustomData.getModuleData(pageContext)`, where `pageContext` is the standard JSP object.

### Getting Module Data

The following example shows how to obtain module-specific data.
CustomData cd = CustomData.getModuleData(pageContext);
String strText = cd.getValue("body.text");
String strType = cd.getValue("body.type");

**Setting Module Data**
The following code demonstrates setting and saving module-specific data.

CustomData cd = CustomData.getModuleData(pageContext);

String strText = request.getParameter("body.text");
String strType = request.getParameter("body.type");
if (strText == null) {
    strText="";
}
if (strType == null) {
    strType="1";
}

cd.setValue("body.text", strText);

Settings Personalization Data
The following code demonstrates how to store user-specific data for a module.

CustomData cd =
    CustomData.getModulePersonalizationData(pageContext);

String strText = request.getParameter("body.text");
if (strText == null) {
    strText="";
}

cd.setValue("body.text", strText);

cd.save();

**Installing the Module Type**
New module types should be installed within a Building Block package, called an installation package. The installation package contains the code, the extension and the deployment information. The package is created by using a tool to .ZIP the files. The folder structure must be preserved, though directory references may be root anchored or relative. This must be consistent throughout the package.

The JSPs corresponding to each module type should be placed under the Web/module directory in the Building Block installation package.
For more information, see the *Blackboard Building Blocks Developers Guide*. 
Manifest Definition

Overview

The manifest is the set of directives the developer provides to *Blackboard Community System* that tells the server what links to render for the extension and where to render them. The manifest provides links in the form of HTTP actions.

Module Type attributes

The following table describes Module Type attributes for the Building Block manifest:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uicreatable</td>
<td>Determines whether a module of this type may be created through the Administrator Control Panel. If uicreatable=true then an Administrator may create a module of this type through the Blackboard Learning System UI.</td>
</tr>
<tr>
<td>ext-ref</td>
<td>External reference name</td>
</tr>
<tr>
<td>title</td>
<td>Title of the Module Type.</td>
</tr>
</tbody>
</table>

The following code is an example of a manifest definition for a module type. The example declares a simple module type, Example Plug-In Module Type.

```xml
<!-- Modules, types, and channels for the portal -->
<module-defs>
  
  <module-type ext-ref="exmp-module" title="Example Plug-In Module Type" uicreatable="true">
    <jsp-dir>module</jsp-dir>
    <jsp>
      <view>view.jsp</view>
      <edit>edit.jsp</edit>
      <admin>admin.jsp</admin>
    </jsp>
  </module-type>

</module-defs>
```
Example

The following code is an example of the View, Admin and Edit pages for a custom module type. This module is designed to display text specified by the Administrator followed by text specified by the user.

**View.jsp**

```jsp
<%@page import="blackboard.portal.external.*" errorPage="/error.jsp" %>
<%@ taglib uri="/bbData" prefix="bbData"%>
<bbData:context>
<%
CustomData cd = CustomData.getModuleData(pageContext);
String strText = cd.getValue("customText");

CustomData cd2 =
CustomData.getModulePersonalizationData(pageContext);
String strUserText = cd2.getValue("userText");
%
<%=strText%>
<br>
<%=strUserText%>
</bbData:context>
```

**admin.jsp**

```jsp
<%@page import="blackboard.portal.external.*" errorPage="/error.jsp" %>
<%@ taglib uri="/bbUI" prefix="bbUI"%>
<%@ taglib uri="/bbData" prefix="bbData"%>
<bbData:context>
<%
CustomData cd = CustomData.getModuleData(pageContext);
String strText = cd.getValue("customText");
if (strText == null)
{
  strText="";
}
%

<bbUI:moduleAdminPage>
<FORM method="post" action="proc_admin.jsp">
<bbUI:step title="Edit Module Text" number="1">
  <bbUI:dataElement label="Text"><textarea wrap =off name="customText" rows=20 cols=55><%=strText%></textarea></bbUI:dataElement>
</bbUI:step>
<bbUI:stepSubmit title="Submit" number="2"/>
```
Localizing the Manifest

Building Blocks may include text that is specific to different languages. Prior to Blackboard Learning System (Release 7), data such as the name of the module or content handlers, were displayed “as-is” from the manifest. Using a simple mechanism to find the language-appropriate text, Building Block developers can provide information for the platform to display the Building Block metadata in a localized fashion. This mechanism, explained below, renders the text in the appropriate language by using a combination of the key from the manifest plus an algorithm to find the appropriate bundle.

The following manifest elements are used by the platform to render information to the end user. To localize them, use a
“bundle key” in the manifest instead of plain text. A bundle key is a simple identifier used by the system to locate the actual text to display. Multibyte characters are accepted in all of these elements.

**Note:** Multibyte characters are accepted in all of these elements.

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>title</td>
<td>Title of Module</td>
</tr>
<tr>
<td>rss-channel</td>
<td>title</td>
<td>Title of rss channel</td>
</tr>
<tr>
<td>module-type</td>
<td>title</td>
<td>label for module type</td>
</tr>
<tr>
<td>module-type</td>
<td>description</td>
<td>long description for module type</td>
</tr>
</tbody>
</table>

The Blackboard Learning System can support multiple languages and includes the ability for end users to create and install their own language packs. To require Building Blocks to support that range of languages is unrealistic. Thus, when rendering the metadata for a Building Block installed on the system, the platform will attempt to find a resource bundle associated with the manifest in the following order:

- User’s current locale
- System’s default locale
- Building Block’s default locale (an optional element in the manifest)
- English (United States)

A file naming convention is used to find the appropriate bundle, `bb-manifest-<locale>.properties`, where `<locale>` is a string in the standard ISO language/country format. The bundles must be in the WEB-INF/bundles directory of the manifest. For example, the following files would be used in a Building Block that supports English (US) and Spanish (Spain):

- WEB-INF/bundles/bb-manifest-en_US.properties
- WEB-INF/bundles/bb-manifest-es_ES.properties

At a minimum, a Building Block should have two bundles to be considered “localizable”, though for backwards compatibility, if no bundles are found, the text in the manifest is returned “as-is”. This allows localization to be optional, and allows old Building Blocks to operate without modification.

The bundle format should follow the standard for Java property bundles; they must be ISO-8869-1 encoded and use Unicode escape sequences for multi-byte characters. Tools such as
Native-to-ASCII (which is part of the standard Java Developer’s Kit) can be used to format the data as needed.

**Note:** This format is required for the bundles that Blackboard will display; the Building Block itself, however, can use whichever bundle format is appropriate.

---

**Localized manifest example**

The following code is an example of a localized manifest definition for a module type.

```
<!-- Modules, types, and channels for the portal -->
<module-defs>

  <module-type ext-ref="exmp-module" title="module-type.title" uicreatable="true">
    <jsp-dir>module</jsp-dir>
    <jsp>
      <view>view.jsp</view>
      <edit>edit.jsp</edit>
      <admin>admin.jsp</admin>
    </jsp>
  </module-type>

</module-defs>
```
Creating Modules

Introduction

Modules, module types, and Channels may all be defined in a Building Block manifest. The following section describes the format required for module types that already exist in the system, such as Channel modules and HTML modules. If a custom module type has already been created and is deployed in the system, this section will also apply.

Manifest Definitions

Modules can be included in a Building Block by creating the right entries in the `bb-manifest.xml` file.

Defining a Module

The following DTD (document type definition) fragment shows the elements used in creating a module.

```xml
<!ELEMENT module (description, ExtraInfo)>  
<!ATTLIST module
type CDATA #REQUIRED
isadmin CDATA #REQUIRED
useraddable CDATA #REQUIRED
isdeletable CDATA #REQUIRED
title CDATA #REQUIRED>
```

The module element defines the basic object. The following table describes the module elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Defines how the module is to be displayed by associating it with a module type. The type attribute must either reference a module type in the current package or a type installed on the target system.</td>
</tr>
<tr>
<td>isadmin</td>
<td>Defines whether the Contents button will appear for Administrators.</td>
</tr>
<tr>
<td>useraddable</td>
<td>Defines whether users may add the module to their personalized view. If false is selected an Administrator must add the module to the layout.</td>
</tr>
<tr>
<td>isdeletable</td>
<td>Defines whether the module can be</td>
</tr>
</tbody>
</table>
Building Blocks Technology (Release 8)            Module Developer Guide

removed from the system. This is useful if the module is part of a Building Block that may include other components such as content handlers or applications.

| title | Defines the human readable name of the module. |

---

**Manifest definition example**

The following code is an example of a manifest definition for an HTML module type.

```xml
<module type="portal/includehtml"
   isadmin="true" useraddable="true"
   isdeletable="true" title="Sample Plug-In Module">

  <description>Sample uploaded module</description>

  <module-groups><module-group id="Everyone" /></module-groups>

  <ExtraInfo>
    <property key="body.text" type="String">This module was installed as part of the sample plugin. It uses the basic 'portal/includehtml' module type.</property>
  </ExtraInfo>

</module>
```

The example declares a simple module type, `portal/includehtml`. The module can be globally edited and added to user-specific layouts.

Extra Info is the data that actually comprises the module.

**Note:** The `body.text` property is used as the text to display by the `portal/includehtml` module type.

The following code is an example of a localized manifest definition for an HTML module type.

```xml
<module type="portal/includehtml"
   isadmin="true" useraddable="true"
   isdeletable="true" title="module.title">

  <description>module.description</description>

  <module-groups><module-group id="Everyone" /></module-groups>

  <ExtraInfo>
    <property key="body.text" type="String">This module was installed as part of the sample plugin. It uses the basic 'portal/includehtml' module type.</property>
  </ExtraInfo>

</module>
```
This module was installed as part of the sample plugin. It uses the basic 'portal/includehtml' module type.

Deployed Module Types

The following list includes the module types shipped with Blackboard Community System that are available to use in the manifest.

<table>
<thead>
<tr>
<th>Type</th>
<th>External Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include HTML</td>
<td>portal/includehtml</td>
<td>Displays the text provided by the user.</td>
</tr>
<tr>
<td>Include URL</td>
<td>portal/includeurl</td>
<td>Displays the URL.</td>
</tr>
<tr>
<td>Parameterized URL</td>
<td>portal/parameterurl</td>
<td>Displays the contents of a URL with form values in the URL.</td>
</tr>
<tr>
<td>RSS Channel</td>
<td>portal/channel</td>
<td>Displays the contents of an RSS channel. The channel can either be a pre-installed channel or one defined in the Building Block.</td>
</tr>
</tbody>
</table>

Defining an RSS Channel module has two parts: defining the module and defining the RSS Channels available to that module.

The Channel definition is as follows:

```xml
<!ELEMENT rss-channel (data-url?)>
<!ATTLIST rss-channel
  ext-ref CDATA #REQUIRED
  title CDATA #REQUIRED
>
<!ELEMENT data-url (#PCDATA)>
```
The data-url element must contain valid, parsed XML. Therefore, characters like ampersand (&) must be encoded properly. For example, the URL

http://example.com/page?argOne=foo&argTwo=rss must be encoded as

http://example.com/page?argOne=foo&amp;argTwo=rss.

Example

```xml
<rss-channel ext-ref="news" title="News">
  <data-url>http://example.com/page?argOne=foo&amp;amp;argTwo=rss</data-url>
</rss-channel>
```
Module-Groups

Each module may have one or more pre-defined groups associated with it. Only users in these groups will be able to access the module. For example, the module can only be made available to the Administrator or Students. Defining the module-group element is optional.

System Administrators may also designate the module-group in the Blackboard Community System. Changes made by Administrators can override pre-defined module-groups.

Extra Info Data

Each module may contain an Extra Info data structure that allows the developer to define the data from the module they would like to save. This section explains the Extra Info for the public module types. Public module types are those that are included in the Blackboard Community System.

Extra Info is a collection of typed properties encoded as a set of XML elements. Custom module types are free to define keys and values.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Name of the property.</td>
</tr>
<tr>
<td>Type</td>
<td>Type descriptor for the data object. “String” is the only supported value.</td>
</tr>
</tbody>
</table>

**Note:** If an Administrator removes a module from the Blackboard Learning System all Extra Info data will be permanently removed.

Exporting Modules

Modules can also be created in the Blackboard Community System through the System Control Panel and then exported.

The end result is a .war file conforming to the Building Block packaging format. This Building Block can be deployed on another implementation of the Blackboard Community System.

For more information see the Blackboard Academic Suite Administrator manual.