Blackboardlearn+

Version 9.0

Building Blocks™ Developer Guide

Blackboard

Publication Date: April, 2009

Worldwide Headquarters	International Headquarters
Blackboard Inc.	**Blackboard International B.V.**
650 Massachusetts Avenue N.W. Sixth Floor Washington, DC 20001-3796	Dam 27 2nd Floor 1012 JS Amsterdam The Netherlands
800-424-9299 toll free US & Canada	
+1-202-463-4860 telephone	+31 20 5206884 (NL) telephone
+1-202-463-4863 facsimile	+31 20 5206885 (NL) facsimile
www.blackboard.com	global.blackboard.com

Table of Contents

Introduction

Building Blocks are an exciting feature in *Blackboard Learn*™. A Building Block is an application created by third party developers that is used to extend the functionality of the core *Blackboard Learn*.

The Building Blocks Software Development Kit is intended to give Building Block authors a thorough overview of the Building Block framework and provide a quick start reference to begin creating Building Blocks.

This document will help developers create Supported Interfaces for Blackboard Learn that are secure, consistent with Blackboard Learn look and feel, integrate with the core application, and operate with external systems. 'Supported Interfaces' are those utilizing Blackboard's published API's pursuant to a valid license, including through the Blackboard Building Blocks Program.

Manual Updates

Please note that the *Building Blocks 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.

Audience

This document is intended for developers creating Building Blocks for Blackboard Learn, the *Blackboard Community System,* and the *Blackboard Content System*. A thorough understanding of Java®, Servlets, and Java Server Pages is assumed. Proficiency in Java server-side programming is also assumed. Additionally, references will be made to the *Building Blocks API Specification Guide* for more details regarding the objects and system services mentioned here.

Quick start

This document explains how to create a Building Block in a step-by-step manner. The following are the high-level steps necessary to create a Building Block:

Step 1 Create the scripts.

Step 2 Create a Building Block manifest file.

Step 3 Create the Building Block package file.

Step 4 Install the Building Block.

Step 5 Register with Blackboard to obtain a vendor ID at http://www.blackboard.com/Communities/Partners/Building-Blocks.aspx.

Building Block Overview

A Building Block is a .zip or .war file that has a structure defined by Blackboard. This structure primarily consists of a file-directory hierarchy containing custom-code tied together by Blackboard Learn manifest. There are many different types of Building Blocks that can be built, including new content type and collaboration tools. This section discusses the definition of a Building Block in depth and offers a number of examples.

In this section

The following topics are included in this section:

What is a Building Block?

A Building Block is simply a set of files installed on the Blackboard Web/app server in a way that is structured so the server has predefined entry points to call upon the functionality of the Building Block. The entry points are Uniform Resource Locators (URLs) that are tracked in Blackboard Learn database and associated with key entities such as content handlers and navigation items. For more information, see the section Using the Building Blocks APIs and Runtime.

Building Block = Web Application

At its very core, a Building Block is simply a Web application, with some supplemental information provided for *Blackboard Learn* to locate the resources within a Building Block. The Java Servlet, specification version 2.2, defines Web applications.

The servlet specification provides a standard way to package the resources in a Web-based application, and a standard deployment framework. For more information see http://java.sun.com/products/servlet/2.2/ .

Entry points = Hyperlinks

Hyperlinks are the entry points and the means by which *Blackboard Learn* calls upon a Building Block. There is not necessarily a one-to-one correspondence between links and JSPs provided by the Building Block. For example, a Building Block may provide scripts that are never rendered as links but instead serve as callbacks for an external system.

Java libraries

Developers may provide supplemental class libraries as part of their application. Deployment of these is defined by the Servlet specification.

Things to Do With a Building Block

What exactly can a developer do with a Building Block? This is one of the first questions that come to mind when looking at the Building Block framework.

While entry points limit where Building Blocks appear in Blackboard Learn user interface, the possible behaviors of a Building Block are not so limited. Building Blocks will appear in a Course Control Panel, via the content editors, or in the Tools or Communication navigation area. While these conventions broadly imply the functionality of a Building Block, they do not constrain it. The following are some examples of uses for a Building Block:

- **Bridge to an External System.** A live hook can be created between *Blackboard Learn* and an external system, through a Building Block. Examples include links to globally hosted databases or locally hosted websites.
- **Content Type.** By creating a Building Block that defines a content handler, a developer can override how the system processes content, allowing them to place custom content types in course and organization content areas, like Course Documents, Books, and Assignments.
- **Student/Instructor Tool.** Tools, such course-specific hooks into a library reservation system, can also be created.
- **Communication Tool.** Communication tools can be developed, for example, a hook to a different chat server.

Developers can also build combinations of any of the above. An example might be a Building Block that bridges Blackboard Learn with an external system for a custom content type. The same Building Block could install a link to tools that provides a user with the ability to manage their account information on the target system.

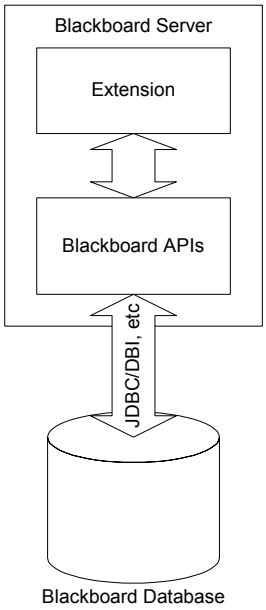
Architecture Examples

In order to illustrate what can be done with a Building Block, two basic architectures for Building Blocks are described, plug-in architecture and bridge architecture.

Plug – in

A plug-in is a Building Block that can be used to provide an independent piece of functionality that does not rely on external servers. An example would be a custom content type that provides an interactive applet in the course environment. This is the most basic Building Block type.

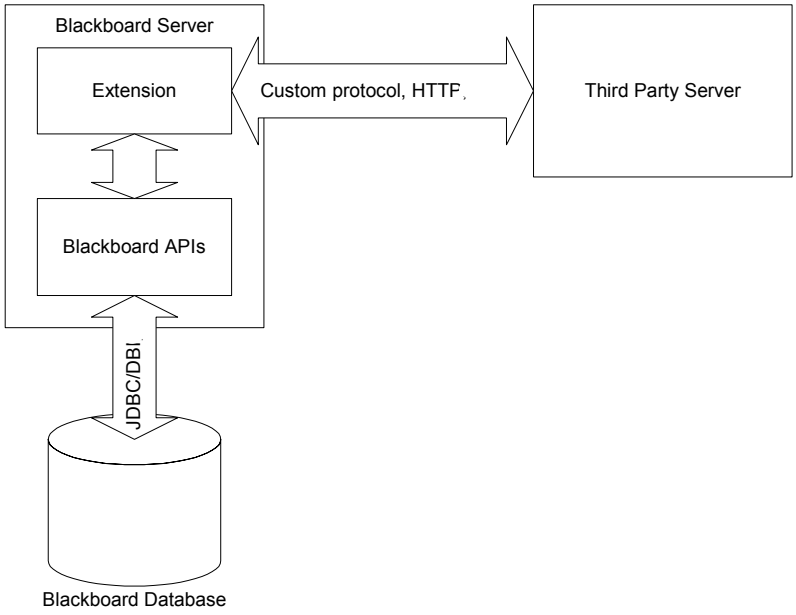
Below is an example of the plug-in architecture.



Bridge

In the bridge architecture, the Building Block can be used to facilitate communication with an external system. There are a variety of ways to implement the communication channel, including using script URLs as callbacks or implementing a custom protocol. An example of this type of Building Block may include a hook to a third-party assessment engine. Bridges can become very complex; some external applications may require complex flow control using a custom protocol.

Below is an example of the bridge architecture.



Building Blocks APIs and Runtime

Blackboard Learn exposes several Java-based APIs to access run-time information. This provides useful functionality in the Building Block. These Building Block APIs are broadly categorized into several distinct sub-sets: Data Objects and Persistence, Session and Context, Gradebook, and Authorization. Although the APIs are provided with Blackboard Learn they can also be used to extend the *Blackboard Community system*.

Note: This section gives a brief overview of the Building Blocks APIs and Runtime that underlies the specific APIs used to create Building Blocks. A more detailed description can be found in the Blackboard Building Blocks: Introduction to the Building Blocks APIs and Runtime.

In this section

The topics in this section include:

- Data Objects and Persistence APIs
- Session and Context, Gradebook, and Authorization APIs

Data Objects and Persistence

Data Objects and Persistence is not a specific Building Block API, rather it is a set of APIs that relate to one another in a common mechanism. This mechanism is used to store the data manipulated in each API. Developers will probably never need to work with the framework directly, but it is valuable to understand its role in storing and retrieving data. For more information on how to use the Persistence framework, see the section Using the Building Blocks APIs and Runtime.

Data Objects

Data objects are encapsulations of data entities in Blackboard Learn and the attached attributes. Data objects within the Building Blocks APIs and Runtime directly map to the entities a user would see represented in the user interface. These objects contain no business logic and act primarily as attribute repositories.

The data objects are independent of any storage or persistence mechanism—in other words, the data objects are completely separate from the containers used to store data.

All data objects are in the system sub-class `blackboard.data.BbObject`. This sub-class contains most of the functionality used by the map-based persistence framework.

Here are examples of available data objects:

- Announcement
- Calendar
- Content
- Context
- Courses
- Discussion Board
- Enrollments
- Gradebook
- Navigation / Course Table of Contents
- Plugins
- Portal Data
- Roles
- Users
- Utility Classes

Persistence Objects

To support several different data stores without making the data objects overly complex, the Blackboard Building Blocks run-time includes sets of Persistence Objects that provide the logic and functionality required to save the data objects into and read from different data stores.

Supported Data Stores

- Oracle®
- Microsoft® SQL® Server

Loaders

Loaders are the objects responsible for reading data from a storage mechanism and converting the data into a live object reference.

Persisters

Persisters are the objects responsible for taking a live object reference and writing the data into a storage mechanism. They offer read-write functions such as insert, update, and delete.

See `blackboard.data.*`

See `blackboard.persist.*`

Using data objects and persisters

At first glance the API may seem complex as it is designed to allow flexible persistence operations. However, many of the steps will be the same in most cases so convenience methods are available to expose the default-configured objects. Most of the objects are automatically initialized in the Blackboard platform's Java environment via a services framework that creates appropriate default instances for all services, loaders, and persister objects. The services framework also maintains the right connection between the persistence objects and the appropriate database for access to the Virtual Installation.

Session and Context, Gradebook and Authorization APIs

The following topic describes the APIs for Session and Context, Gradebook, Authorization, and Authentication.

Session and Context

All requests to the application are associated with a Session. This object retains information about the user and their authentication status. Context is not a specific object but represents information about the current request that is not readily identified by Session, such as the course being accessed.

See `blackboard.platform.session.BbSessionManagerService`.

Gradebook

Some Building Blocks need to write data into the Gradebook. A simple API is provided to allow the creation of line items and the recording of scores into the line items for users.

See `blackboard.data.Gradebook.*`

See `blackboard.persist.Gradebook.*`

Authorization

Authorization comprises two parts: verifying that the current user has appropriate access to perform a specific action and verifying that the code being executed is also trusted. The user authorization is handled by the `AccessManager` object. Code verification is performed by the JVM. Building Blocks are required to include in the manifest a brief security descriptor that identifies permissions required to operate.

See `blackboard.platform.security.AccessManager`.

Authentication

Authentication is handled automatically by the login subsystem of *Blackboard Learn*. If a Building Block must ensure that authentication has occurred, it may call `Session.isAuthenticated()`. If this method returns false `HttpAuthManager.sendLoginRedirect()` may be called to initiate a login sequence. A utility method in `PlugInUtil` is also available to wrap this functionality.

See `blackboard.platform.session.`

Using the Building Blocks APIs and Runtime

This section describes some general information about the Building Blocks APIs and Runtime that underlies the specific APIs used to create Building Blocks.

Note: A more detailed description of the APIs and Runtime can be found in the Blackboard Building Blocks: Introduction to the Building Blocks APIs and Runtime.

In this section

The following topics are included in this section:

- Top-level Package Structure
- The Blackboard Data Model
- Strongly-types Enumerations
- Persistence Services

Top-Level Package Structure

This section describes the top-level structure of the Java API.

Top-level Packages

The following table describes the top-level packages in the Building Blocks APIs.

Package Name	Description
`bb-platform`	Persistence framework package. This package contains the core elements in the Building Blocks APIs and Runtime, including the `BbPersistenceManager` class. Its sub-packages Announcement, Assessment, Calendar, Course, Registry, Task, and User, define the Persister and Loader interfaces available for each data bean. These interfaces make up the core of the persistence framework.
`bb-ws-context`	This package contains the `ContextWS` class, which provides the initial methods required for session creation and is therefore required to be used before any other webservices can be utilized..
`bb-ws-util`	Utility package.

Blackboard Data Model

The data bean classes in the various sub-packages of `bb-platform` provide almost complete coverage of the data that is tracked by Blackboard Learn.

Data sub-packages

The data beans are grouped into the following sub-packages:

Data sub-package	Classes contained
`announcement`	Announcements in the system.
`calendar`	Contains the `CalendarEntry` class that is used by the Calendar subsystem.
`class-use`	Classes for attributes, links, objects, receipts, and validations.
`content`	Contains classes for creating and manipulating course content.
`course`	Contains classes for core learning system concepts such as courses, groups, course classification and course membership. Also contains the `ButtonStyle` class.
`discussionboard`	Contains classes relating to the Discussion Board.
`gradebook`	Contains classes relating to the Grade Center.
`navigation`	Contains classes relating to the Course Table of Contents and packages.
`role`	Contains classes relating to roles and portals.
`user`	Contains core classes pertaining to users and other user information, such as an Address Book entry.

It is important to note that the data model is independent of any storage or persistence mechanism, meaning the data model is completely de-coupled from the persistence services. All data beans in the model inherit directly or indirectly from the class `BbObject`. `BbObject` can be thought of as the base persistent object class. `BbObject` is made up of an ID, created date, and modified date. Data beans are limited to contain only a set of attributes and getter and setter methods for those attributes.

Strongly-Typed Enumerations

All enumerations in the Building Block APIs are implemented using a pattern that provides for compile-time type checking of element membership in an enumeration. In this pattern the primary enumeration class includes the following properties:

- Only private constructors
- Static final instances of itself are defined for each element in the enumeration

Class

For convenience, the enumeration classes are frequently declared as inner classes of a more general class that provides for the scope and context. For example, class `Course` contains an inner class called `Pace`. Given this class, methods like `course.setPace (Course.Pace.INSTRUCTOR_LED)` can be called. To switch off of the elements in an enumeration, the "==" operator can be used. The example below typed enumeration example demonstrates this point.

```
Course.Pace pace = myCourse.getPace();
if ( pace == Course.Pace.INSTRUCTOR_LED ) {
 /* handle case one */
}
else if ( pace == Course.Pace.SELF_PACED ) {
 /* handle case two */
}
else {
 throw new InternalError(
     "unknown enumeration element encountered: " +
pace.toString() );
}
```

Tips and tricks

It is good form to explicitly handle each case in the enumeration—the code should not "fall through" with an else and assume a certain state unless not all cases in the enumeration require explicit handling. This reduces code readability and introduces the possibility of an error if an enumeration is widened at a later date. Blackboard encourages all developers to use this pattern instead of switching on statically defined integers or strings.

Persistence Services

Access to the persistence services of the Building Blocks APIs and Runtime originates with the `BbPersistenceManager` class. For the remainder of this document, the term “persistence manager” will be used to denote an instance of the class `blackboard.persist.BbPersistenceManager`.

Container

A persistence manager is associated with a particular container. A container can be thought of as a mechanism that provides persistent storage for objects such as a database or file system. Blackboard persistence APIs work transparently against Oracle and SQL Server database back ends.

Loaders and persisters

The services offered by the Building Blocks APIs and Runtime are defined in a set of implementation independent Java interfaces. An instance of the appropriate implementation class for a given interface must be obtained from a persistence manager. These interfaces are divided into two main categories: Loaders and Persisters. Loaders offer read-only operations on the data model such as looking up an object by its ID or searching for a list of objects based on given criteria. Persisters offer read-write functions such as insert, update, and delete. The following code shows how to look up an implementation of a loader for Courses from a pre-initialized instance of `BbPersistenceManager` called `bbPm`.

Example of performing a loader lookup:

```
CourseDbLoader loader = (CourseDbLoader) bbPm.getLoader(
CourseDbLoader.TYPE );
//access CourseDbLoader.loadByXXX() methods
```

Blackboard Look and Feel

When creating a Building Block it is useful to the end user to make the Building Block's interface fit seamlessly into the Blackboard interface. There are two that can help accomplish this: the Blackboard tag libraries, used to render common visual elements, and icons supplied as part of the Building Block.

In this section

The topics in this section include:

- Blackboard Tag Libraries
- Icons

Building Blocks Tag Library

The *Building Blocks Tag Library* is a Java class library used to abstract user interface components as XML-like tags that are evaluated by Java classes. It is published in a separate manual that can be found on Behind the Blackboard. All of the tags and associated information are included.

The tag library architecture was standardized in the JSP 1.1 specification from Sun© Microsystems.

More information about the JSP specifications can be found at:

Java Servlet Specification, version 2.2. http://www.javasoft.com/products/servlet/2.2/

Java Server Pages (JSP) Spécification, version 1.1.
http://www.javasoft.com/projects/jsp/1.1/

While it is not required to use the Blackboard tags, it is highly recommended, as it helps ensure a seamless experience for the user. It also helps to ensure that the Building Block can evolve as the Blackboard UI becomes available in different contexts, such as Wireless Application Protocol (WAP) or text-only versions for high accessibility.

Icons

Icons are the visual cues associated with a Building Block in the application. A Building Block must provide an icon to display within the course context.

List Item icon

The List Item icon is displayed when the Building Block entry point is displayed in a list with other system entry points. An example is the Tools page within a course. The List Item icon must be 32 x 32 pixels.

Writing Content Building Blocks

Content Building Blocks are Building Blocks that interact in the content areas of a course. They allow custom content types to be placed in the course or organization content areas, such as Course Documents, Assignments, and Books. Course Documents are also referred to as Content Items.

In this section

The following topics are included in this section:

Entry Points

Entry points are defined for each content handler included in the manifest. Content handlers are used to render a Building Block's links for creating custom content types. Information on the manifest can be found in the *Blackboard Learn* Manifest topic of this document.

The scripts provided in the Content Handler definition are the entry points of a content Building Block. They are rendered in two contexts: from the Add Custom page (create action) and from the Content Area editor pages (modify and remove actions).

Create action

Create action is rendered from within the content editing areas in a course through the **Add Custom** link. The links are not rendered directly. The display page includes logic to generate the appropriate parameters and direct the browser to the create script.

The create script must accept the following parameters:

Parameter	Description
`content_id`	The ID, in string form, of the parent object for the new document. This must be set on the new document object.
`course_id`	The ID, in string form, of the containing course.

Modify/Remove action

The Modify and Remove actions are rendered from within the content editing areas in a course and in line with content items that may be edited.

The remove action is not directly invoked as a Blackboard-provided script does most of the processing. The Blackboard script deletes the actual Course Document. The execution of the remove script provided by the Building Block allows the Building Block to clean up or clear any external resources it may be holding.

Additionally, the modify script should support the following parameters:

Parameter	Description
`content_id`	The ID, in string form, of the content object to delete.
`course_id`	The ID, in string form, of the containing course.

Using Content Items

The key ability of a content Building Block is being able to place content into the Content Item hierarchy associated with a course. There are several components of the persistence API to facilitate this.

Create a content item

Creating a `content` object is as simple as instantiating the data object and setting its attributes. The following code sample shows how to create and save a content item.

```
// retrieve the Db persistence manager from the persistence
service
BbPersistenceManager bbPm =
BbServiceManager.getPersistenceService().
  getDbPersistenceManager();
// create a course document and set all desired attributes
Content content = new Content();
content.setTitle( "Sample Item" );
FormattedText text = new FormattedText(
  request.getParameter( "text" ), FormattedText.Type.HTML );
content.setBody( text );
content.setContentHandler( "resource/x-smpl-type1" );
// ... set additional attributes ...
// these attributes of content require valid Ids... create and
set them
Id courseId = bbPm.generateId( Course.DATA_TYPE,
  request.getParameter( "course_id" ) );
Id parentId = bbPm.generateId( Content.DATA_TYPE,
  request.getParameter( "content_id" ) );
content.setCourseId( courseId );
content.setParentId( parentId );
// retrieve the content persister and persist the content item
ContentDbPersister persister = (ContentDbPersister)
  bbPm.getPersister( ContentDbPersister.TYPE );
persister.persist( content );
```

In the example above, the text of the content item is entered through the `Content.setBody()`. This method accepts a FormattedText object which encapsulates

both the text of the content item as well as the formatting of that text. Presently, three types of formatting are supported: HTML, Plain Text, and Smart Text (Smart Text uses some of the basic features of HTML without having to enter HTML code).

Note: Templates can be included to make some details of the content formatting easier, such as content.url to reference uploaded files.

Using ID Objects

IDs play a key role within the Blackboard API -- they serve to uniquely identify objects within the Blackboard system allowing for future retrieval and/or modification. In order to be system unique, Ids encompass several pieces of information, including key data, object type and other identifying information. IDs cannot be directly instantiated and must be created using BbPersistenceManager.generateId().

A string representation of an Id value may be obtained via Id.toExternalString(). This is useful when it is necessary to pass an Id value from one process to another and the object itself cannot be passed—for example, passing an Id on a URL from one JSP page to another.

Note: toString() is not appropriate for this purpose as it is used to provide additional debugging information.

Save the document object

Once the object has been created it is saved by getting a reference, through the `BbPersitenceManager`, to a persister and invoking its persist() method.

The following is an example of saving a Course Document:

```
ContentDbPersister persister = (ContentDbPersister)
  bbPm.getPersister( ContentDbPersister.TYPE );
persister.persist( content );
```

Note: The process is identical if the document is new or was loaded from the database. The persistence objects determine what action to take.

Loading a Document

In addition to persister objects the developer will work with loaders to create objects from stored data. In most cases, a parameter passed from the modify action script will be referenced. This is typically encoded as a string. The `BbPersistenceManager` can be used to translate the string to a valid ID object for use in a load operation.

Note: IDs are actual objects in the persistence framework and must be generated by the PersistenceManager.

The following code demonstrates loading a document.

```
ContentDbLoader loader =
  (ContentDbLoader) bbPm.getLoader( ContentDbLoader.TYPE );
Id contentId = bbPm.generateId( Content.DATA_TYPE,
  request.getParameter( "content_id" ) );
Content content = loader.loadById( contentId );
```

Content Item Types

The Blackboard Content API includes many different data objects. The base content class is called Content. All sub-classes of Content (CourseDocument, ExternalLink, etc.) represent native content types understood by the Blackboard system. Use of these sub-classes should be limited to times when the desire is to create a native Blackboard content item. If creating your own content type (the typical reason for creating a content Building Block), then the Content class should be used as shown in the examples above.

Because the ContentDbLoader can return any of the content data objects, it is safer to treat all objects returned by it as the base class (Content). This avoids possible object casting problems. If you must cast to another content type, it is safer to use getDataType() or instance of to first validate the type before trying to cast.

Context Passing

Context passing is useful to System Administrators when they are implementing Building Blocks that require content from *Blackboard Learn* to generate a URL.

The context passing APIs allow *Blackboard Learn* to pass data to URLs requiring that data in a query string. To see an example of this type of URL, simply look at the URL for a course in *Blackboard Learn*. The last part of the URL is `url=/bin/common/course.pl?course_id=<unique_id>` where the <unique_id> is a variable. It is variables such as this that can be passed using the context passing APIs.

Developers may want to include parameters in links embedded in a course document. Without a dynamic rendering framework some additional actions are required. *Blackboard Learn* provides the URL with a feature for passing contextual data through URL templates. Templates are URLs provided by users that contain placeholders for data that will be inserted at render time.

There are several variables that can be used to embed information in the HTML included for rendering. The advantage to using templates is that the information does not have to be hard coded and is thus more portable from course to course. This is particularly relevant in course copy and import/export actions.

Course Document expansion

If `@X@` are used as the delimiters the variables will be expanded when rendered in a Course Document. The benefit of this approach is that there is no intermediate step between the display of the template and navigating the link.

Deferred expansion

A slightly different syntax can be used if Course Document expansion is not desired Instead, have the link reference the URL `/webapps/blackboard/launch_external.jsp` and include a URL template in the `url` parameter. Below is an example of a deferred expansion URL:

```
/webapps/blackboard/launch_external.jsp?encrypt=y&url=http://exa
mple.com/page?
uid=@X@user.user_id@X@
```

The benefit of this approach is that there is an intermediate step to invoke the `launch_external.jsp`. This allows the developer to specify that the URL should be encrypted. This step creates a temporary key that external systems (with live database access to Blackboard Learn), can use to verify requests.

Run-time expansion

Developers can also have provided templates expanded at runtime by the `Session` object. The templates are the same, but instead are processed in the context of a method call.

See `blackboard.platform.session.BbSession.encodeTemplateURL()`

Variables

The following variables may be used in URL templates:

Variable	Value	Example
course.batch_uid	The external identifier for a course. Accepts multibyte characters.	ABC123ABC
course.id	The simple abbreviation for a course.	BIO101
course.role	A user's role in the current course or organization. Accepts multibyte characters.	student
course.url	The base URL for all files referenced in the course.	/courses/1/BIO101/
content.id	The internal identifier for content in a course.	_23_1
content.url	The base URL for files associated with a given piece of content.	/courses/1/BOB101/content/_x_y
system.site_id	The host name of the current virtual installation.	bb_bb60
request.id	Unique identifier for the current session formatted as a 32 character hex digit string based on the time of access. Guaranteed to be unique across the system.	35853280-A77A-11D8-83D5-9CAA2FE644E1
request.return	The first non-null value of: The "return" parameter in the query string The "return" parameter in the URL the URL of the referring page.	http:/localhost/webapps/bbgs-bbqa-context-bb_bb60/tool_1/tool.jsp?course_id=_2_1
session.id	Hexadecimal string representing the session identifier for the current session.	8f14e45fceea167a5a36dedd4bea2543
user.batch_uid	The external identifier for a user. Accepts multibyte characters.	123-45-6789

Variable	Value	Example
user.id	The current Username. Accepts multibyte characters.	jsmith
user.role	The current user's System Role.	System Roles C- Course Administrator U- Guest N- None O- Observer Y- Community Administrator R- Support Z- System Admin H- System Support A- User Administrator
user.institution_role	The Role ID of the current user's Primary Institution Role. Accepts multibyte characters.	student
user.primary_institution_role	The Role ID of the current user's Primary Institution Role (same as institution_role). Accepts multibyte characters.	student
user.secondary_institution_role	A comma-delimited list of all of the user's secondary Role IDs.	student,faculty
membership.role	A user's role in the current course or organization. Accepts multibyte characters.	Course/Organization Roles B- Course Builder/Organization Builder G- Grader/Grader U- Guest/Guest P- Instructor/Leader S- Student/Participant T- Teacher's Assistant/Assistant

Example One:

When given the user jdoe, in the course CS114, the URL template `user_id=@X@user.user_id@X@&course_id=@X@course.course_id@X@` would expand as: `user_id=jdoe&course_id=CS114`.

Example Two:

A Building Block includes a Java applet that reads a file from the server to display math equations. The file is stored in the course document's file repository. The following applet tag could automatically generate the appropriate URL without hard coding the location.

Below is an example of using templates in Course Document HTML:

```
<applet code="vendor.AppletClass" archive="/webapps/vend-
plgn/applet.jar">
    <param name= "download" value=
"@X@content.url@X@/file.mml">
</applet>
```

Example Three:

The `Session` object can be used to encode parameters directly in the plug-in script handlers. Below is an example of using session to encode a template URL:

```
BbSession bbSession =
    BbServiceManager.getSessionManagerService().getSession(
request );
String encodedUrl =
    bbSession.encodeTemplateUrl( request,
request.getParameter("target") );
```

Encoding the template URL can be used to pass information to external systems or the tool. The `batch_uid` property on both user and course is used in integration scenarios and typically maps to an ID maintained by an external system, for example the primary key used within the SIS.

Note: When performing deferred expansion, the developer must be able to include the appropriate IDs (such as Course ID) to ensure that the context information is available.

Interacting with the Grade Center

Content Building Blocks may need to interact with the Grade Center. This is done through the Gradebook API, which exists in the packages `blackboard.data.gradebook` and `blackboard.persist.gradebook`.

Line items

This interface allows the developer to create line items and attach scores to them. Additionally, the developer can link both the line items and scores to external analysis programs, via the `attemptHandlerURL` property of the `blackboard.data.gradebook.Lineitem` object. This is particularly useful if the Building Block is a bridge to an external assessment engine.

Scores

Score objects represent the actual graded outcome of a student's interaction with a "grade-able" resource. Even though an individual score may be calculated from one or more attempts, only a single score value is exposed for each line item in this version of the API.

Basic Usage

Using the Gradebook APIs typically comprises the following steps:

Step 1	Create a Lineitem and set the appropriate properties, including the courseId.
Step 2	Persist Lineitem.
Step 3	Store the LineitemID reference for later use.
Step 4	When a Student interacts with a "gradeable" resource, create a Score object and set the appropriate properties, including the stored Lineitem Id.
Step 5	Persist Score.

Using the File System

Every Course Document has an associated file store. The developer can use the file store to save data for their Building Block.

Note: These files are not tracked by the system and will be deleted when the content object is removed.

Directories

The `FileSystemService` gives access to directory spaces within the content areas.

The directories retrieved from the `FileSystemService` must be treated as unrelated. That is, any reliance on paths existing outside of the provided directory will likely result in errors in subsequent releases.

Warning: Do not rely on the layout of directories returned from the FileSystemService as it will change in subsequent releases.

See `blackboard.platform.filesystem.FileSystemService`

Writing Tool Building Blocks

Writing a Tool Building Block can be a very open-ended task, and can touch a number of issues. This section will review the basics of a Tool in the Blackboard context and the various points of interaction in the system.

Note: This section makes the assumption that the reader is familiar with the Web-based navigational structure of Blackboard Learn.

Applications

In Blackboard Learn all of the links that may be associated with tools in a system are related via the Application entity. For example, the different links for the Announcements tool are connected via an Application object.

The end user manages applications via the System Control Panel and the Course Control Panel. For example, open **Manage Tools** on the Control Panel. The tools installed by default are managed differently than tools installed by Building Blocks.

Building Blocks may define one or more applications in the manifest. For the Building Block this means that all of the links associated with that Building Block will be managed as a single unit.

Entry Points

There are four pre-defined entry points for a Building Block to use in the context of a course. The entry point used depends on the use case.

- **Control Panel.** The link is displayed in the Course Tools section of the Course Control Panel.
- **System Administration Panel.** The link is displayed in the System Tools section of the System Control Panel.
- **Communications.** The link is added to the set of tools displayed in the default Communications navigation area of the course. This area is located in the Course Menu.
- **Course Tools.** The link is added to the set of tools displayed in the default Tools navigation area of the course. Tools are located in the Course Menu.
- **User Tools.** The link is added to the Tools box available on the community tab areas.

Course Tools and User Tools are somewhat arbitrary. Once an Application is defined for a set of links Instructors can create any number of access points to that application. For example, if a Building Block installs a link that will appear in the Tools area of a course, an Instructor can create a new course area and point it directly to the entry point of the Building Block.

The entry points are determined by the type attribute on the link element in the manifest.

Communication Tools vs. Course Tools

Functionally, there is no difference in the API for use by either type of Building Block. The distinction is made solely for the details the developer wants to apply for the tool.

Writing Content System Building Blocks

Beginning with *Blackboard Content System* (Release 2.3), it is possible to write Building Blocks with features specific to the Content System. Developers have the option of either creating Building Blocks that require the Content System to be installed, or creating Building Blocks that integrate with the Content System when it is available but do not require it.

Installation

In addition to specifying the required Blackboard platform version in the Building Block manifest, Content System Building Blocks should specify the minimum required version of the Content System in a `csversion` sub-element of the `requires` element. When a Content System Building Block is installed, the version of the Content System currently installed is checked against the version specified in the Building Block manifest. If the currently installed version is older than the version in the manifest, the Building Block will not be installed, and the Administrator will receive an error message stating the required Content System version. If the currently installed version is the same as the version in the manifest or newer, the Building Block will be installed successfully.

If the Content System is not installed at all, the `ifMissing` attribute of the `csversion` element is checked. When set to “fail” the Building Block will not be installed, and the Administrator will receive an error message stating that the Content System must be installed to use the Building Block. When set to “warn” the Building Block will be installed, but the Administrators will receive a message stating that some features of the Building Block may be unavailable unless the Content System is installed. This can be used to create Building Blocks that integrate with the Content System when it is available, but can also be used with only Blackboard Learn or *Blackboard Community System* installed.

Applications

Writing a Content System Building Block is very similar in most respects to writing a Tool Building Block. As with Tool Building Blocks, Content System Building Blocks are grouped within an Application object. Tool Building Blocks and Content System Building Blocks can be freely intermixed within a single application definition, or can be defined in separate applications.

Entry Points

The Content System defines six additional entry points for use by Content System Building Blocks.

- **Content System Tools.** The link is added to the Tools box in the folder view of the Content Collection menu, and the Tools menu available from the shortcut view of the Content Collection menu.
- **Content System Action Bar.** The link is added to the drop-down menu at the end of the Action Bar on folder listing pages in the Content Collection. Action Bar Building Blocks act on the files and folders currently selected in the folder listing.
- **Modify Content File.** The link is added to the menu of options that appears when a user clicks **Modify** on a file in the Content Collection.
- **Modify Content Folder.** The link is added to the menu of options that appears when a user clicks **Modify** on a folder in the Content Collection.
- **Manage Portfolio.** The link is added to the menu of options that appears when a user clicks **Manage** on a Portfolio on the My Portfolios page.

- **My Portfolios.** The link is added to the My Portfolios box in the folder view of the Content Collection menu, and the Portfolios menu available from the shortcut view of the Content Collection menu.

Execution Contexts

Content System Tools and My Portfolios Building Blocks are system-wide tools. Links added in those areas by Building Blocks will not have any additional HTTP request parameters appended beyond what is specified by the developer in the Building Block manifest. Links in the Content System Action Bar, Modify Content File, Modify Content Folder, and Manage Portfolio areas, however, are meant to operate on one or more Content System objects, specified by an HTTP request parameter. The `blackboard.cms.servlet` package contains classes that encapsulate the use of these request parameters, and can be used by Building Blocks developers to determine the appropriate object or objects to act on.

Example 1: Content System Action Bar Building Blocks

This JSP can be used as the target of a Content System Action Bar link. It will print out a list containing the path names of the files and folders that were selected in the folder listing where it was used.

```
<%@page import="blackboard.cms.servlet.CSActionRequest" %>
<%@page import="java.util.Iterator" %>
<%@taglib prefix="bbData" uri="/bbData" %>
<bbData:context>
<!-- Print out a list of the selected files and folders -->
<ul>
<%
  CSActionRequest actionReq =
    new CSActionRequest( request, response, application );
  List selectedFiles = actionReq.getSelectedPaths(); // returns
a list of path names
  for ( Iterator i = selectedFiles.iterator(); i.hasNext(); )
  {
    String fileName = (String) i.next();
%>
  <li><%= fileName %>
<%
  }
%>
</ul>
</bbData:context>
```

Example 2: Modify Content File/Folder Building Block

This JSP can be used as the target of a Modify Content File or Modify Content Folder link. It will print out the path name of the file being modified.

```
<%@page import="blackboard.cms.servlet.CSModifyEntryRequest" %>
<%@taglib prefix="bbData" uri="/bbData" %>
<bbData:context>
<%
  CSModifyEntryRequest modifyEntryReq = new
CSModifyEntryRequest( request );
  out.println( modifyEntryReq.getPath() );
%>
</bbData:context>
```

Example 3: Manage Portfolio Building Block

This JSP can be used as the target of a Manage Portfolio link. It will print out the title of the Portfolio being managed.

```
<%@page import="blackboard.cms.portfolio.Portfolio" %>
<%@page import="blackboard.cms.servlet.CSManagePortfolioRequest"
%>
<%@taglib prefix="bbData" uri="/bbData" %>
<bbData:context>
<%
  CSManagePortfolioRequest portfolioReq =
    new CSManagePortfolioRequest( request );
  Portfolio portfolio = portfolioReq.getPortfolio();
  out.println( portfolio.getTitle() );
%>
</bbData:context>
```

General Development Tasks

The following section describes the higher-level tasks of authentication and authorization that are useful in Building Blocks.

In this section

The following topics are included in this section:

- Authenticating Users
- Authorizing Users

Authenticating Users

Authentication is performed by Blackboard Learn, and will rarely, if ever, get called directly from a plug-in. For more information see the topic on Authentication.

Checking authentication

Authentication status is queried by referencing a `Session` object and checking the `isAuthenticated()` method.

Below is an example of checking authentication:

```
BbSession bbSession = sessionService.getSession( request );.
.
if (! bbSession.isAuthenticated())
{
     HttpAuthManager.sendLoginRedirect(request,response);
     return;
}

See blackboard.platform.session.BbSessionManagerService
```

Authorizing Users

User authorization is performed through `AccessManagerService`.

Authorization example

This example demonstrates how to use the method `isUserInSystemRole()`. The roles used in the `sysAllowedRoles` array are defined in the enumeration `blackboard.data.user.User.SystemRole`.

The following is an authorization example:

```
//roles that are allowed access
User.SystemRole sysAllowedRoles[] = {
     User.SystemRole.SYSTEM_ADMIN,
     User.SystemRole.SYSTEM_SUPPORT,
     User.SystemRole.ACCOUNT_ADMIN,
     User.SystemRole.COURSE_CREATOR
};
//get access manager
AccessManagerService accessManager = (AccessManagerService)
     BbServiceManager.lookupService( AccessManagerService.class
);
//perform check
if (!accessManager.isUserInSystemRole( request, sysAllowedRoles
)) {
     HttpAuthManager.sendAccessDeniedRedirect(request,response);
     return;
}
```

Note: The supported roles are enumerated in the class User.SystemRole. The utility method HttpAuthManager.sendAccessDeniedRedirect() sends the user to a page that displays a message if authorization fails.

Authorization utility methods

There are utility methods on `PlugInUtil` that authorize predefined sets of roles based on the default *Blackboard Learn* behavior.

Method	Description
authorizeForCourseControlPanel	Verifies that the user associated with the current session is enrolled in the current course as an Instructor, Teaching Assistant, or Course Builder.
authorizeForSystemAdmin	Verifies that the current user has a System Administrator role of System Administrator, System Support, Account Admin, or Course Creator.

Creating a Building Block

This section describes the areas outside of programming that developers should take into account when building a Building Block for the Building Blocks APIs and Runtime.

Note: All of the code described in this section is included in the sample Building Block package delivered as part of the Building Blocks Software Developer Kit.

In this section

The following topics are included in this section:

- Development environment
- Deciding what to build
- Debugging the Building Block

Development Environment

Developers may create a Building Block with any Java development tools they are familiar with. The Building Block may simply contain a few JSP files or it may contain custom class libraries developed to encapsulate more advanced functionality. If class libraries that rely on the Blackboard classes are going to be developed then the `CLASSPATH` of the development tool must be set up.

CLASSPATH

For many Building Blocks, the development environment does not need to be configured unless custom classes are being created. Many Building Blocks will be able to provide their functionality exclusively through JSP scripts. The `<blackboard>/bbservices/systemlib/bb-persistence.jar` needs to be added to the `CLASSPATH` variable used by the development environment when creating custom class libraries that use Blackboard Learn Java classes.

Utility Libraries

Blackboard Learn ships with several libraries that may be useful for general-purpose development.

- **Gnu-regex-1.0.8.** A package for parsing regular expressions.
- **Xerces-1.4.3.** The Apache XML parser. Contains facilities for DOM and SAX parsing and support for name spaces and document type definition (DTD) validation.
- **Xerces-1.2.0.** The Apache XML parser. This is an older version shipped for backward compatibility.
- **Xerces-1.0.3.** The Apache XML parser. This is an older version shipped for backward compatibility.

Java Database Connectivity (JDBC)

In this version of the Building Blocks API and Runtime direct database access is possible though not encouraged. All of the required database operations are accomplished with the persistence APIs. These wrap the details of database interaction, allowing the programmer to focus on data. The one exception is obtaining a Connection object to control transactions.

Additional libraries

Additional libraries may be provided in the installation package either as expanded class files or as a bundled .jar file. Class files go into the `/WEB-INF/classes` directory and .jar files go into the `/WEB-INF/lib` directory.

Deciding What to Build

The Building Block sample that is provided is a simple Building Block that presents a form field for text entry. The example does not provide much practical use as this feature is already available in the core platform but it does illustrate most of the components needed to develop for a Building Block.

The following steps are dependent on the type of functionality being provided. They are each discussed in detail in this section.

Step 1 Build the content handler actions.
Step 2 Build the links.
Step 3 Build a configuration script.
Step 4 Build a remove script.

Build the Content Handler Actions

When building a custom content type, HTTP actions for each content handler defined in the Building Block manifest must be provided. These scripts are always invoked in a content context, which means that the URL contains parameters that define the current course and content object.

When the create action is called the content ID passed is that of the parent folder. When creating a Course Document it must be associated with a folder.

When the other modify and remove actions are called the content ID refers to the item to modify or delete. Thus an action such as load from database is performed based on the content ID.

Build Links

Links are different from content handlers and configuration scripts in that they are not clearly defined by http actions. Instead, they are defined with a link element and a type attribute. This is because in the tool-oriented areas of the application (course tools and communications) a single entry is provided. The type attribute determines where the application should render the link.

Build a Configuration Script

All Building Blocks that are required to provide a configuration script are identified in the manifest by the configuration HTTP action element. The script is not required to do anything except display a message. In the example a simple screen that allows us to select the items to query will be added.

Configuration data for the plug-in may be stored in the file system. The directory may be obtained via `PlugUtil.getConfigDirectory()`.

Build a Remove Script

By providing a remove script, the developer can perform any necessary cleanup when the Building Block is removed from Blackboard Learn. For example, the Building Block may need to un-register itself from external systems. This is not an opportunity to delete all associated content or file system components; that is handled by Blackboard Learn. The remove script is not mandatory. The script is summoned by a `RequestDispatcher.include()` call and is not expected to display any user interface or interact with the user.

Note: There are certain limitations to building a Remove Script. The Remove Script must not attempt to alter the client response. This means it cannot write any HTML, which may produce unpredictable results since a different script generates the receipt. It also cannot attempt to set an alternate HTTP status code which will result in an IllegalAccessException error.

Debugging the Building Block

Blackboard Learn (Release 9) is designed to run against the Java 2 SDK 5.0.

Debugging support is automatically provided by Java via the Java Virtual Machine Debugging Interface (JVMDI).

Building Block XML Packaging Format

Packaging a Building Block includes creating a .ZIP file that conforms to the Servlet 2.2 Web Application Archive (.war) specification and includes additional data used by Blackboard Learn server to install the Building Block's entry points.

In this section

The following topics are included in this section:

- Web Archive Overview
- URLs
- Blackboard Manifest
- Packaging the Building Block

Web Archive Overview

A Web archive file is a .ZIP file conforming to the definition provided in Section 9 of the Sun Java Servlet Specification, version 2.2.

Sub-directories

Below is a brief description of the sub-directories that may be included in the WEB-INF directory or WAR file layout.

Directory/File	Description
`WEB-INF/classes/`	Expanded class files.
`WEB-INF/lib/`	.jar files included for the application. These may include libraries that are developed or third party libraries included as utilities.
`WEB-INF/web.xml`	The Web app deployment descriptor (required by the Servlet 2.2 specification).
`WEB-INF/bb-manifest.xml`	The Blackboard package definition (required by Blackboard Learn package specification).
`/`	The root of the Building Blocks. With the exception of WEB-INF developers may organize this directory as they see fit. Typically, this will include the Building Blocks JSP scripts, images, and so forth.

URLs

Once the application has been called up many Web applications rely on the ability of supplemental scripts to provide processing. This is typically done by HTTP redirects or Request Dispatches. Care must be taken in the construction of URLs as the evaluation of the URL is dependent on the method used.

Obtaining a URL to the Building Block

Due to the dynamic deployment environment with Virtual Installations the Building Block should not hard code any root-anchored, self-referencing URLs. Instead, one of the utility methods on `blackboard.platform.plugin.PlugInUtil` should be used to obtain the URL reference.

Redirection vs. request dispatch

In the servlet specification there are two methods to return an alternative response to a client, redirection and request dispatch.

- Redirection involves an actual HTTP response code sent to the client to request a different page.
- A request dispatch allows a script on the server to invoke the functionality of another script. Under the servlet specification there are different semantics for URLs in each case.

Redirects

Redirects are generated via the `HttpServletResponse.sendRedirect()` method, which is exposed by the Java Servlet API. A redirection sends an HTTP 302 to the client resulting in the client requesting a new target URL. If a relative URL is provided as the argument to `sendRedirect()` then the servlet container translates the call to a fully qualified URL. URLs to Building Blocks are typically rendered to the client in the form `/webapps/<extension_id>/<extension script>`.

Note: The Web server (Apache on UNIX, IIS on Windows) uses a proxy agent to delegate requests to the servlet engine. The proxy agent strips /webapps from the URL thus the developer needs to ensure that all root-anchored Universal Resource Identifiers (URIs) include /webapps, for example, /webapps/smpl-plugin/test.jsp.

Request dispatch

Request dispatch is a technique for delegating requests on the server and is defined in the Servlet specification. This is a more confusing case.

The Servlet specification defines an object called a `RequestDispatcher`. There are two methods: `forward()` and `include()`. A Request Dispatcher is obtained from a number of sources, all of which require a Uniform Resource Identifier (URI). The URI is evaluated relative to the Web application's context. That is, if a URI such as `/some.jsp` is provided it will be evaluated against the Web application's root, which is equivalent to /`webapps/<extension id>`. Thus the full URL would be `/webapps/<extension>/some.jsp`.

Consider the following sets of scripts in a Building Block:

- /webapps/smpl-plugin/create.jsp
- /webapps/smpl-plugin/modify.jsp
- /webapps/smp-plugin/include.jsp

If either `create.jsp` or `modify.jsp` requires access to `include.jsp` as an included resource the following code snippet would be used:

```
RequestDispatcher rd =
application.getRequestDispatcher("include.jsp");
rd.include( request, response );
```

Note: Application, request, and response are JSP built-in objects.

Encoded URLs

When generating URLs for use with `sendRedirect()` or display in an `<A>` element it is good form to use the session object to encode the URL. This will encode any session or context information that may need to be passed back to the server on subsequent requests.

Blackboard Learn Manifest

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

Manifest Definitions

There are several main link types defined by the manifest:

- **Basic Information and Configuration.** Used to render a Building Block's configuration and remove links.
- **Content Handler.** Used to render a Building Block's links for creating custom content types.
- **Application.** Used to render Building Blocks in course tools and course communication links.
- **Module.** Data that is processed by a module type to render a module in the community home page.
- **Module Types.** Custom logic used to display modules in the community home page.
- **RSS Channels.** A specific data entity used by both the RSS Channel module type and RSS modules that reference channel definitions.

Note: For Module, Module Types, and RSS Channel definition, see the Module Developers' Guide for more information. They are not covered in detail in this section.

Community System Modules, Module Types, and RSS Channels can also be defined in the manifest.

Localizing the Manifest

Building Blocks may include text that is specific to different languages. Prior to Blackboard Learn (Release 7), data such as the name of the Building Block 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.

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

Element	Attributes	Description
name	value	Name of the plug-in
description	value	A description of the plug-in
description	value	A description of the vendor
content-handle	name	A name for custom content types.
application name	name	name for application
application	description	long description for application
link	name	link label
link	description	link long description
module	title	Title of Module
rss-channel	title	Title of rss channel
module-type	title	label for module type
module-type	description	long description for modules type

Blackboard Learn 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.

Basic Information and Configuration

The root element for a Building Block's manifest is always <plugin>. The basic information for a Building Block is defined in the first few child elements of the <plugin> element.

Element	Description
Name	A name to display for the Building Block in the management screens. Required. Maximum length: 50 characters. Accepts multibyte characters.
Handle	A unique string value used to associate with the Building Block. The handle, in combination with the Vendor Id, is considered to be the unique identifier for this Building Block. Required. Maximum length: 32 characters.
Vendor	Root element for information about the Building Block vendor. Required.
Description	Human readable description for the Building Block. Displayed to administrators in the "view components" management screen. Required. Maximum length: 255 characters. Accepts multibyte characters.
Default-locale	Language to use when the Building Block does not support the current user's locale.
Requires	Root for the Building Block prerequisites. Contains <bbversion> and <csversion> elements. Required.
Bbversion	Element to define dependency on a specific Blackboard version. Value is of the form `major.minor.patch.build`. Only two numbers are required, but for precision, the first three are recommended.
csversion	Element to define dependency on a specific *Blackboard Content System* version. The value has the same format as in the <bbversion> element. Not required.
ifMissing	Attribute of <csversion> element that determines whether the package can be installed without the Content System present. Possible values are "fail" and "warn". Not Required. Default value: warn.
Http-actions	Container element for configuration and removal data. Required.
Config	URI for invoking configuration information for the Building Block. Not Required. Maximum length: 512 characters
Remove	URI for invoking removal action for the Building Block. The script must not perform any action for the user interface (writing HTML, setting status codes). It's intended only as a notification mechanism so the Building Block can perform any required cleanup. Not Required. Maximum length: 512 characters

Element	Description
Id	Vendor ID. Arbitrary string chosen by the developer to identify institution or organization that authored the Building Block. Child of <vendor> element. Required. Maximum length: 4 characters.
Name	Vendor name. Child of <vendor> element. Required. Maximum length: 50 characters. Accepts multibyte characters.
url	Vendor URL. Child of <vendor> element. Not Required. Maximum length: 255 characters.

Defining a content handler

Use the `<content-handler>` container element, and define a name, handle, icon, and HTTP-actions.

Element	Description
Name	A name to display in the Add Other drop-down list in the Instructor page editors. Accepts multibyte characters.
Handle	A unique string value used to connect a content database entry with its corresponding handler. The syntax is not strictly defined; however, by convention a MIME-like syntax is commonly used.
Http-Actions	Defines the collection of entry points for editing content created by this handler.
Create	Script called to create a content object. The script must accept `course_id` and `content_id` arguments. The `content_id` argument for creation references the parent folder. Maximum length is 512 characters. This URL must be relative to the root of the Web application.
Modify	The script called to modify the content created by this handler. The script must accept a `course_id` and `content_id` arguments. The `content_id` argument is the object to modify. Maximum length is 512 characters. This URL must be relative to the root of the Web application.
Remove	The script called to follow up on deletion of content. The script must accept a `content_id` argument. Note that the actual deletion is performed by Blackboard Learn prior to invoking this script; thus there is no direct access to the object. This is called to allow cleanup by the Building Block. Maximum length is 512 characters. This URL must be relative to the root of the Web application.
Icons	Collection of icons to display to the user. Icon should be 32 x 32 pixels to display properly in the course/organization environment.
Toolbar	Reserved. Child element of Icons.
List item	Not used; child element of Icons. *Blackboard Learn*

Example

The following example defines a simple content handler.

```
<content-handlers>
      <content-handler>
            <name value="content-handler.handle"/>
            <handle value= "resource/x-smpl-type1"/>
            <http-actions>
                  <create value="ch1/create.jsp"/>
                  <modify value="ch1/modify.jsp"/>
                  <remove value="ch1/remove.jsp"/>
            </http-actions>
            <icons>
             <toolbar value="/images/add_ch1.gif"/>
             <listitem value="/images/icon.gif"/>
            </icons>
      </content-handler>
</content-handlers>
```

Defining an application

Links should appear inside an <application> definition.

Name	Description
Application-defs	The container for multiple application definitions.
Application	The entity that defines the grouping of links.
Handle	Simple string identifier to uniquely identify the application on the system. This string is combined with the vendor ID string.
Type	The type for the application. This must be one of “course”, “shared” or “system”. A course application is only used in the course or organization environment. A “system” application can expose an entry point in the system context. A “shared” application may be used in courses, or at the system level.
Name	The user-friendly name of the application. Maximum length: 64 characters. Accepts multibyte characters.
Small-icon	Reserved.
Large-icon	Reserved.
Description	A description of the application. Maximum length: 3900 characters. This field is not currently displayed to end users. Accepts multibyte characters.
Can-allow-guest	Flag indicating that the tool can accept anonymous (or guest) users.
Links	The collection of links exposed by the containing application.

Name	Description
Link	A single link within the application.
Type	Type of the link. This must be one of: tool, communication, course_tool, user_tool, system_tool, cs_tool, cs_action, cs_modify_file, cs_modify_folder, cs_manage_portfolio, or cs_my_portfolios. An application may contain any number of links of any type. For example, the developer may choose to provide two separate links to a tool within the “Tools” area of the course.
Name	Name to display for the link. Maximum length: 255 characters. Accepts multibyte characters.
URL	Relative path of the tool. This must be relative to the web application root (but not root-anchored). Maximum length: 255 characters.
Description	Description of the link to display. Maximum length: 3900 characters. This is not currently displayed to end users. Accepts multibyte characters.
Icons	The container for icon definitions.
Listitem	The icon to display in the list mode for course navigation areas.

Example

```
    <application-defs>
      <application handle="sampleapp" type="course" use-
ssl="false"
       name="application.name" can-allow-guest="true"
       small-icon="" large-icon="">
        <description >application.description</description>
        <links>
          <link>
            <type value="tool"/>
            <name value="tool.name"/>
            <url value="links/tool1.jsp" />
            <description value="tool.description" />
            <icons>
              <listitem value="/images/icon.gif"/>
            </icons>
          </link>
      </application>
    </application-defs>
```

Declaring security

A section of the manifest is set aside to declare permissions that are required to run the Building Block. Refer to the *Building Blocks API Specifications* for more information on what permissions are required for specific operations.

The XML format corresponds closely to the format used in standard Java policy files. This includes type attributes to define the Java class name for the requested permission, a “name” attribute and an “actions” attribute. The “name” and “actions” attributes are defined by the Permission object in the core Java API. The exception is for mnemonic names for Blackboard-defined permissions or commonly requested Java permissions. The following mnemonic names are defined:

- **Persist**. Permission required to load or persist a data object. The name is the type of action; the allowed actions are load and persist.
- **Runtime**. Wrapper for a Java runtime permission. Name and actions are defined in the Java platform API.
- **Socket**. Wrapper for a Java socket permission. Name and actions are defined in the Java platform API.

Name	Description
Permissions	Container for permissions.
Permission	Individual permission to apply.
Type	Type string, either a mnemonic or fully qualified class name.
Name	Name of the permission.
Actions	Actions required for the permission to perform correctly.

Example

```
<permissions>
  <permission type="persist" name="Content" actions="persist"/>
  <permission type="socket" name="*.blackboard.com" actions="connect"/>
</permissions>
```

Complete Manifest

The following sample is a complete manifest and is included with the sample plug-in.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<manifest>
  <!-- core extension information -->
  <plugin>
    <name value= "Sample Plugin"/>
    <handle value= "plgnhndl"/>
    <description value= "This plugin is a sample."/>
    <version value= "1.0.0.1"/>
    <requires>
       <bbversion value="6.3.0"/>
        <csversion value="2.3.0" ifMissing=”warn”>
    </requires>
    <vendor>
      <id value="smpl"/>
      <name value="Sample Plugin Vendor"/>
      <url value="http://www.samplevendor.com/" />
      <description value="The description of the sample vendor goes here." />
    </vendor>
    <http-actions>
      <config value="admin/config.jsp"/>
      <remove value="admin/remove.jsp"/>
    </http-actions>

    <!-- Custom content types defined by this extension -->
    <content-handlers>
      <content-handler>
        <name value="Blackboard Sample Plug-in: HTML Block"/>
        <handle value= "resource/x-smpl-type1"/>
        <http-actions>
          <create value="ch1/create.jsp"/>
          <modify value="ch1/modify.jsp"/>
          <remove value="ch1/remove.jsp"/>
        </http-actions>
        <icons>
          <toolbar value="/images/add_ch1.gif"/>
          <listitem value="/images/icon.gif"/>
        </icons>
      </content-handler>
      <content-handler>
        <name value="Blackboard Sample Plug-in: Text Block"/>
        <handle value= "resource/x-smpl-type2"/>
        <http-actions>
          <create value="ch2/create.jsp"/>
          <modify value="ch2/modify.jsp"/>
          <remove value="ch2/remove.jsp"/>
        </http-actions>
        <icons>
          <toolbar value="/images/add_ch2.gif"/>
          <listitem value="/images/icon.gif"/>
        </icons>
      </content-handler>
    </content-handlers>

    <!-- Tools defined by this extension -->
    <application-defs>
```

```
      <application handle="sampleapp" type="course" use-ssl="false"
name="Sample Application" can-allow-guest="true"
       small-icon="" large-icon="">
        <description lang="en_US">Application installed as part of the sample
plugin</description>
        <links>
          <link>
            <type value="tool"/>
            <name value="Sample Tool 1"/>
            <url value="links/tool1.jsp" />
            <description value="The description of Sample Tool 1." />
            <icons>
              <listitem value="/images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="communication"/>
            <name value="Sample Communication Tool 2"/>
            <url value="links/tool2.jsp?mode=73" />
            <description value="The description of Sample Communication Tool
2." />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="course_tool"/>
            <name value="Plugin Control Panel Tool"/>
            <url value="links/controlPanelTool.jsp" />
            <description value="Demonstrates adding tools to the course control
panel." />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="user_tool"/>
            <name value="Plugin User Tool"/>
            <url value="links/tool2.jsp?mode=73" />
            <description value="Demonstrates adding tools available to all
users." />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="system_tool"/>
            <name value="Plugin Admin Panel Tool"/>
            <url value="links/tool2.jsp?mode=73" />
            <description value="Demonstrates adding tools to the system
administration tool." />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="cs_action"/>
            <name value="CS Action"/>
            <url value="actionbar.jsp"/>
            <description value="Sample CS Action Bar Plug-in"/>
            <icons></icons>
          </link>
          <link>
```

```
            <type value="cs_tool"/>
            <name value="CS Tool"/>
            <url value="tool.jsp"/>
            <description value="Sample CS Tool Plug-in"/>
            <icons></icons>
          </link>
          <link>
            <type value="cs_modify_file"/>
            <name value="CS Modify File"/>
            <url value="modifyfile.jsp"/>
            <description value="Sample CS Modify File Plug-in"/>
            <icons></icons>
          </link>
          <link>
            <type value="cs_modify_folder"/>
            <name value="CS Modify Folder"/>
            <url value="modifyfolder.jsp"/>
            <description value="Sample CS Modify Folder Plug-in"/>
            <icons></icons>
          </link>
          <link>
            <type value="cs_manage_portfolio"/>
            <name value="CS Manage Portfolio"/>
            <url value="manageportfolio.jsp"/>
            <description value="Sample CS Manage Portfolio Plug-in"/>
            <icons></icons>
          </link>
          <link>
            <type value="cs_my_portfolios"/>
            <name value="CS My Portfolios"/>
            <url value="myportfolios.jsp"/>
            <description value="Sample CS My Portfolios Plug-in"/>
            <icons></icons>
          </link>
        </links>
      </application>
    </application-defs>

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

      <module-type ext-ref="smpl-module" title="Sample Plug-in Module Type"
uicreatable="true">
        <jsp-dir>module</jsp-dir>
        <jsp>
          <view>view.jsp</view>
          <admin>admin.jsp</admin>
        </jsp>
      </module-type>

      <rss-channel ext-ref="gamenews" title="Game News">
        <data-url>http://p.moreover.com/cgi-
local/page?c=Computer%20games%20news&amp;o=rss</data-url>
      </rss-channel>

      <module type="portal/channel" isadmin="true" useraddable="true"
isdeletable="true" title="Sample Channel Module">
        <description>Sample channel module. This module accesses the RSS
channel installed with this plug-in.</description>
        <ExtraInfo>
          <property key="channel.id" type="String">smpl-gamenews</property>
        </ExtraInfo>
      </module>
```

```
      <module type="portal/includehtml" isadmin="true" useraddable="true"
isdeletable="true" title="Sample Plug-In Module">
        <description>Sample uploaded module</description>
        <ExtraInfo>
          <property key="body.text" type="String">This module was installed as
part of the sample plugin. It uses the
           basic 'includetext' module type.</property>
        </ExtraInfo>
      </module>

    </module-defs>

    <!-- code permissions required for proper operation -->
    <permissions>
      <permission type="attribute" name="user.givenname" actions="get,set"/>
      <permission type="persist" name="Content" actions="persist"/>
      <permission type="socket" name="*.blackboard.com" actions="connect"/>
    </permissions>
  </plugin>

</manifest>
```

Localized Manifest

The following sample is a complete manifest for a localized Building Block.

Note: The manifest should use an encoding appropriate for the data contained in the file. UTF-8 is recommended, but if the manifest contains no multibyte characters, 1SO-8869-1 and US-ASCII are appropriate as well.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<manifest>
  <!-- core extension information -->
  <plugin>
    <name value= "plugin.name"/>
    <handle value= "plgnhndl"/>
    <description value= "plugin.description"/>
    <default-locale value="en_US"/>
    <version value= "1.0.0.1"/>
    <requires>
       <bbversion value="7.0.0"/>
    </requires>
    <vendor>
      <id value="smpl"/>
      <name value="Sample Vendor"/>
      <url value="http://www.samplevendor.com/" />
      <description value="plugin.vendor.description" />
    </vendor>
    <http-actions>
      <config value="admin/config.jsp"/>
      <remove value="admin/remove.jsp"/>
    </http-actions>

    <!-- Custom content types defined by this extension -->
    <content-handlers>
      <content-handler>
        <name value="plugin.content-handler1.name"/>
        <handle value= "resource/x-smpl-type1"/>
        <http-actions>
          <create value="ch1/create.jsp"/>
          <modify value="ch1/modify.jsp"/>
          <remove value="ch1/remove.jsp"/>
        </http-actions>
        <icons>
          <toolbar value="/images/add_ch1.gif"/>
          <listitem value="/images/icon.gif"/>
        </icons>
      </content-handler>
      <content-handler>
        <name value="plugin.content-handler2.name"/>
        <handle value= "resource/x-smpl-type2"/>
        <http-actions>
          <create value="ch2/create.jsp"/>
          <modify value="ch2/modify.jsp"/>
          <remove value="ch2/remove.jsp"/>
        </http-actions>
        <icons>
          <toolbar value="/images/add_ch2.gif"/>
          <listitem value="/images/icon.gif"/>
```

```
        </icons>
      </content-handler>
    </content-handlers>

    <!-- Tools defined by this extension -->
    <application-defs>
      <application handle="sampleapp" type="course" use-ssl="false"
name="plugin.application1.name" can-allow-guest="true"
       small-icon="" large-icon="">
        <description lang="en_US">plugin.application1.description</description>
        <links>
          <link>
            <type value="tool"/>
            <name value="plugin.application1.tool.name"/>
            <url value="links/tool1.jsp" />
            <description value="plugin.application1.tool.description" />
            <icons>
              <listitem value="/images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="communication"/>
            <name value="plugin.application1.communication_tool.name"/>
            <url value="links/tool2.jsp?mode=73" />
            <description
value="plugin.application1.communication_tool.description" />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="course_tool"/>
            <name value="plugin.application1.course_tool.name"/>
            <url value="links/controlPanelTool.jsp" />
            <description value="plugin.application1.course_tool.description" />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="user_tool"/>
            <name value="plugin.application1.user_tool.name"/>
            <url value="links/tool2.jsp?mode=73" />
            <description value="plugin.application1.user_tool.description" />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
          <link>
            <type value="system_tool"/>
            <name value="plugin.application1.system_tool.name"/>
            <url value="links/tool2.jsp?mode=73" />
            <description value="plugin.application1.system_tool.description" />
            <icons>
              <listitem value="images/icon.gif"/>
            </icons>
          </link>
        </links>
      </application>
    </application-defs>

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

```
      <module-type ext-ref="smpl-module" title="plugin.sample-module.title"
uicreatable="true">
        <jsp-dir>module</jsp-dir>
        <jsp>
          <view>view.jsp</view>
          <edit>edit.jsp</edit>
          <admin>admin.jsp</admin>
        </jsp>
      </module-type>

      <rss-channel ext-ref="gamenews" title="plugin.rss-
channel.gamenews.title">
        <data-url>http://p.moreover.com/cgi-
local/page?c=Computer%20games%20news&amp;o=rss</data-url>
      </rss-channel>

      <module type="portal/channel" isadmin="true" useraddable="true"
isdeletable="true" title="plugin.sample-channel.module.title">
        <description>plugin.sample-channel.module.description</description>
        <ExtraInfo>
          <property key="channel.id" type="String">smpl-gamenews</property>
        </ExtraInfo>
      </module>

      <module type="portal/includehtml" isadmin="true" useraddable="true"
isdeletable="true" title="plugin.sample-includehtml-module.title">
        <description>plugin.sample-includehtml-module.description</description>
        <ExtraInfo>
          <property key="body.text" type="String">plugin.sample-includehtml-
module.extra-info.text</property>
        </ExtraInfo>
      </module>

    </module-defs>

    <!-- code permissions required for proper operation -->
    <permissions>
      <permission type="persist" name="Content"
actions="create,modify,delete"/>
      <permission type="attribute" name="user.authinfo" actions="get"/>
    </permissions>
  </plugin>

</manifest>
```

Packaging the Building Block

To install a Building Block on Blackboard Learn the various pieces must be collected into a file, called an installation package, which contains all the code, the Building Block, and deployment information.

Providing a Deployment Descriptor

The servlet specification defines a deployment descriptor to be used in Web applications. Building Blocks are deployed as Web applications; therefore, a Web deployment is required. For the purposes of deploying a Building Block this can be as minimal as possible. This deployment descriptor is defined by the Servlet specification. It is an XML file that must be named WEB-INF/web.xml. Blackboard supports the Servlet 2.3 specification. The contents of this are not controlled or used directly by Blackboard; any valid Servlet 2.3 web.xml file is acceptable. This includes defining custom tag libraries, servlets, servlet filters, and MIME mappings. Refer to the servlet specification (http://java.sun.com/servlets) for more information.

The following Deployment Descriptor is an example of what is included in the sample.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
 <display-name>PlugInManager</display-name>
 <taglib>
  <taglib-uri>/bbUI</taglib-uri>
       <taglib-location>/WEB-INF/config/taglibs/bbUI.tld</taglib-location>
 </taglib>
</web-app>
```

Zipping the Building Block

To create the package use 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.

Note: If additional .jar files are included the Building Block must be packaged with compression turned off. For example, using the jar tool with the -0 flag will package the files together without compression. Failure to do so may result in unpredictable behavior.

Migrating a Building Block

Developers may migrate a Building Block from Blackboard 5.x to *Blackboard Learn* (Release 6.x). This section reviews the steps for migration and offers several examples.

Note: This section does not apply for Blackboard Learn (Release 7). There is a seamless migration for Building Blocks from Blackboard Learn (Release 6.x to Blackboard Learn (Release 7).

Migration steps

The following steps outline how to migrate a Building Block from Blackboard 5.x to *Blackboard Learn* (Release 6.0). Release 6.0 Building Blocks will work with Release 6.1.

Step 1	Change the manifest to include security information.
Step 2	Change the manifest to include application definitions.
Step 3	Remove references to Lightweight objects and associated load/persist objects. See `blackboard.data.*` and `blackboard.persist.*`
Step 4	Update content methods to use the `ContentDbLoader` and `ContentDbPersister`. See `blackboard.data.persist.content.*`.
Step 5	Update Java Server Pages (JSPs) to use the <context> tag in the Blackboard Data tag library
Step 6	Update servlets or non-JSP controller logic to set/release Context accordingly

See the Building Blocks API Specification Guide for more information.

Examples

Step 1 Example

The following XML allows the Building Block to store Content objects, and make socket connections to addresses in the blackboard.com domain.

```
<permissions>
  <permission type="persist" name="Content" actions="persist"/>
  <permission type="socket" name="*.blackboard.com" actions="connect"/>
</permissions>
```

Step 2 Example

Adding an application definition can be done by wrapping existing link definitions in an `<application-defs>` and `<application>` tag.

```
<application-defs>
  <application handle="sampleapp" type="course" use-ssl="false"
   name="Sample Application" can-allow-guest="true"
   small-icon="" large-icon="">
    <description lang="en_US">Application installed as part of the sample
     plugin</description>
    <links>
      <link>
        <type value="tool"/>
        <name value="Sample Tool 1"/>
        <url value="links/tool1.jsp" />
        <description value="The description of Sample Tool 1." />
        <icons>
          <listitem value="/images/icon.gif"/>
        </icons>
      </link>
    </links>
  </application>
</application-defs>
```

Step 3 Example

Heavyweight objects are a superset of lightweight objects and thus are structurally compatible. To remove references to Lightweight objects replace the variable declarations. For example, the following code:

```
LwCourse course;
```

becomes:

```
Course course;
```

Step 4 Example

`CourseDocumentDbLoader` and `ExternalLinkDbLoader` should no longer be used. Instead, load all content types with `ContentDbLoader`.

Previously, the code was written as:

```
CourseDocumentDbLoader courseDocumentLoader =
  (CourseDocumentDbLoader) bbPm.getLoader(
CourseDocumentDbLoader.TYPE );
CourseDocument courseDoc =
  (CourseDocument)courseDocumentLoader.loadById( contentId );
```

In *Blackboard Learn* (Release 9), the code should be written as follows:

```
ContentDbLoader courseDocumentLoader =
  (ContentDbLoader)bbPm.getLoader( ContentDbLoader.TYPE );
CourseDocument courseDoc =
  (CourseDocument)courseDocumentLoader.loadById( contentId );
```

Note: ContentDbLoader methods return Content objects; thus the results must be cast to the desired object type.

Step 5 Example

The context tag is required to initialize any request processing that deals with the correct Virtual Installation.

Add a reference to the Data tag library:

```
<%@ taglib uri="/bbData" prefix="bbData"%>
```

Wrap all logic for the page inside <context> tags. Note that because of the syntax for declaring tag library references, there will always be a prefix for the tag that the user defines, e.g., <bbData:context>.

```
<bbData:context>
<!-- your code goes here -->
. . .
</bbData:context>
```

Step 6 Example

When writing non-JSP classes, such as a servlet, the Context must be manually initialized and released, as shown below:

```
      try {
        //get services
        ctxMgr = (ContextManager)BbServiceManager.lookupService(
ContextManager.class );
        Context ctx = ctxMgr.setContext(request);

        //process . . .

      }
      catch ( Exception e ) {
        //handle error...
      }
      finally {
        if( ctxMgr != null ) {
          ctxMgr.releaseContext();
        }
      }
```

Note: Make sure ContextManager.releaseContext() is performed in a finally block. This will ensure that context gets released.

Advanced Development Issues

Currently, Building Blocks are additions to Blackboard Learn. They can be as simple or as complex as desired. For some types of Building Blocks, however, significant complexity may extend beyond the scope of the Web application container. To meet a wide array of needs the Blackboard APIs are designed to be both portable and flexible. There are numerous possibilities for Building Blocks, especially with the assistance of Blackboard Technical Services. Some of these possibilities are discussed in this section.

Shipping additional libraries

The Web application portion of the servlet specification states that Web applications may ship both custom classes and .jar files. One caveat is that if a library contains a duplicate of a class contained in Blackboard Learn class path, unpredictable behavior may result in the Building Block. The reason is that the class loader used for the Web application will find and load the system's version of the custom class before it loads the one found in the custom class created in the Building Block.

Third-party class libraries

The following third-party class libraries are included in Blackboard Learn class path:

- Gnu-regexp-1.0.8.jar
- Xerces-1.4.3.jar
- Gnu-getopt-1.0.8.jar

Off-line tools

Most of the Blackboard APIs are meant to be portable and can be used outside of Blackboard Learn server runtime to create offline tools. Not all system services are available, but much of the content runtime is set up to allow data objects to be created and manipulated in external programs.

Troubleshooting

Questions and problems may arise during the creation of a Building Block. The following section enables developers to deal with some of the most common issues.

Threading and synchronization

Though not exhaustive, the following list represents some of the possible problems in the multi-threaded application server environment.

- Many threads can safely access one persistence manager instance simultaneously. However, multiple threads are *not* allowed to simultaneously use the same database connection.
- Due to efficiency issues the typed list classes are *not* synchronized. If a list is cached that may be modified by multiple threads get a synchronized version of it through the Java collections APIs.
- In general, if there are more threads than database connections in the pool the threads may block while trying to obtain a connection. The order in which the threads obtain a connection is dependent on the connection pool implementation and may not be first-in-first-out, as expected.

Installation issues

Problem: Installing the Building Block results in the error “Error registering plugin”, with additional error information about “invalid entry CRC.”

Solution: Do not specify compression when creating a .war file that includes .jar files that are already compressed.

Classpath Issues

Problem: At run-time the JSP compiler fails to locate classes.

Solution: Any .jar files needed to run the Building Block must be in the WEB-INF/lib directory of the Building Block. An example would be a third party library.

Classes that are compiled from source and not put in .jar files should be in WEB-INF/classes. That is the standard directory within the Web/app structure.

Web Application Issues

Problem: Accessing any link for the Building Block results in a “Page Not Found” message.

Solution: Ensure that there is a valid `web.xml` file in WEB-INF directory. The Building Block will not be registered as a Web application if there is not a `web.xml`. Additionally, if the Universal Resource Identifier (URI) specified in the link is not correctly mapped in the `web.xml` the servlet container will not be able to resolve the URL.