



Blackboard

Blackboard Building Blocks™ on the Microsoft .NET Platform Developer Guide

Release 7
Blackboard Learning System™
Blackboard Community System™

Date Published: October 12, 2005

Copyright © 2005 by Blackboard Inc. All rights reserved.

Blackboard, the Blackboard logo, Blackboard Academic Suite, Blackboard Learning System, Blackboard Learning System ML, Blackboard Community System, Blackboard Transaction System, Blackboard Building Blocks, and Bringing Education Online are either registered trademarks or trademarks of Blackboard Inc. in the United States and/or other countries. Intel and Pentium are registered trademarks of Intel Corporation. Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States and/or other countries. Sun, Solaris, UltraSPARC, and Java are either registered trademarks or trademarks of Sun Microsystems, Inc. in the United States and/or other countries. Oracle is a registered trademark of Oracle Corporation in the United States and/or other countries. Red Hat is a registered trademark of Red Hat, Inc. in the United States and/or other countries. Linux is a registered trademark of Linus Torvalds in the United States and/or other countries. Apache is a trademark of The Apache Software Foundation in the United States and/or other countries. Macromedia, Authorware and Shockwave are either registered trademarks or trademarks of Macromedia, Inc. in the United States and/or other countries. Real Player and Real Audio Movie are trademarks of RealNetworks in the United States and/or other countries. Adobe and Acrobat Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Macintosh and QuickTime are registered trademarks of Apple Computer, Inc. in the United States and/or other countries. WordPerfect is a registered trademark of Corel Corporation in the United States and/or other countries. Crystal Reports is a trademark of Crystal Decisions in the United States and/or other countries. WebEQ is a trademark of Design Science, Inc. in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners. Patents pending.

© 2005 Blackboard Inc. All rights reserved. Made and printed in the USA.

No part of the contents of this manual may be reproduced or transmitted in any form or by any means without the written permission of the publisher, Blackboard Inc.

Worldwide Headquarters

Blackboard Inc.
1899 L Street, NW, 5th Floor
Washington, DC 20036-3861 USA
800-424-9299 toll free US & Canada
+1-202-463-4860 telephone
+1-202-463-4863 facsimile
www.blackboard.com

International Headquarters

Blackboard International B.V.
Keizersgracht 106
1015 CS Amsterdam
The Netherlands
+31 20 5206884 (NL) telephone
+31 20 5206885 (NL) facsimile
global.blackboard.com

Table of Contents

Introduction.....	4
Introduction to .NET	5
Extension Overview	9
What is a Building Block?.....	10
Things to Do With a Building Block	11
Architecture Examples.....	12
Building Blocks APIs and Runtime	14
Data Objects and Persistence	15
Session, Portal and Authorization APIs.....	17
Using the Building Blocks APIs and Runtime.....	18
Top-Level Package Structure	19
Blackboard Data Model.....	20
Strongly-Typed Enumerations	21
Blackboard Look and Feel	22
Building Blocks Web Controls	23
Icons	31
Writing Content Extensions.....	32
Entry Points.....	33
Using Course Content.....	35
Context Passing	37
Interacting with the Gradebook	39
Writing Tool Extensions	40
General Development Tasks.....	42
Authenticating Users	43
Authorizing Users	44
Building a Building Block.....	45
Development Environment.....	46
Deciding What to Build	47
Building Block XML Packaging Format.....	49
Web Archive Overview	50
URLs	51
<i>Blackboard Learning System</i> Manifest.....	52
Packaging the Building Block	58
Advanced Development Issues	59
Troubleshooting	60
Module Building Blocks	61

Introduction

Overview

Building Blocks™ is an exciting feature in the *Blackboard Learning System™*, *Blackboard Community System™* and *Blackboard Transaction System™*. A Building Block is an application created by third party developers that is used to extend the functionality of the core *Blackboard Learning System*, *Blackboard Community System*, or *Blackboard Transaction System*. Until recently, it was only possible to write these Building Blocks using the Java Programming Language. With Blackboard version 6.1, it is now possible to create Building Blocks using the Microsoft® .NET™ Framework, which allows developers to write Building Blocks using any of nearly 30 programming languages.

The Building Blocks on the Microsoft .NET Platform Software Development Kit is intended to give Building Block .NET authors a thorough overview of the extension framework and provide a quick start reference to begin creating Building Blocks using Microsoft .Net technology.

This document will help developers create Supported Interfaces for the *Blackboard Academic Suite* that are secure, consistent with the *Blackboard Academic Suite* 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.

This document is not a full tutorial of .NET, but is intended to serve as a quick start that helps developers to get started in understanding Building Blocks on the Microsoft .NET Platform.

Audience

This document is intended for developers creating Building Blocks with .NET for the *Blackboard Learning System*. A thorough understanding of .NET Framework, ASP.NET and Web Services is assumed. Proficiency in C# programming is also assumed. Additionally, references will be made to the *Building Blocks.NET API Specification Guide* for more details regarding the objects, web controls and web 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** An ASP.NET web application.
 - Step 2** Create a Building Block manifest file.
 - Step 3** Create the extension package file.
 - Step 4** Install the extension.
-

Introduction to .NET

Overview

Microsoft's .NET initiative is a set of Microsoft software technologies for connecting information, people, systems, and devices. It enables a high level of software integration through the use of XML Web services—small, discrete, Building Block applications that connect to each other, as well as to other, larger applications over the Internet.

There are four major subdivisions of technologies within the .NET initiative: .NET Web Services, .NET Smart Clients, .NET Servers, and .NET Developer Tools. The .NET Web Services consist of a set of services provided by Microsoft and other companies, that provide a range of functionality, including authentication and calendaring; Smart Clients consist of Microsoft application software and operating systems that enable users to utilize Web Services from any location; .NET Servers are the Microsoft operating systems that provide the best server infrastructure for deploying and managing next generation applications, and the .NET Developer Tools make it possible to easily develop a wide range of applications using a range of languages and tools. The remainder of this document concentrates on the .NET Developer Tools.

Microsoft's .NET Developer tools, which consist of Microsoft's .NET Framework and Microsoft's Visual Studio .NET, allow developers to easily build their own XML Web Services, as well as a large range of other types of applications, using any modern programming language. The .NET Framework is an integral Windows component that supports building and running the next generation of applications and XML Web services. The .NET Framework is designed to fulfill the following objectives:

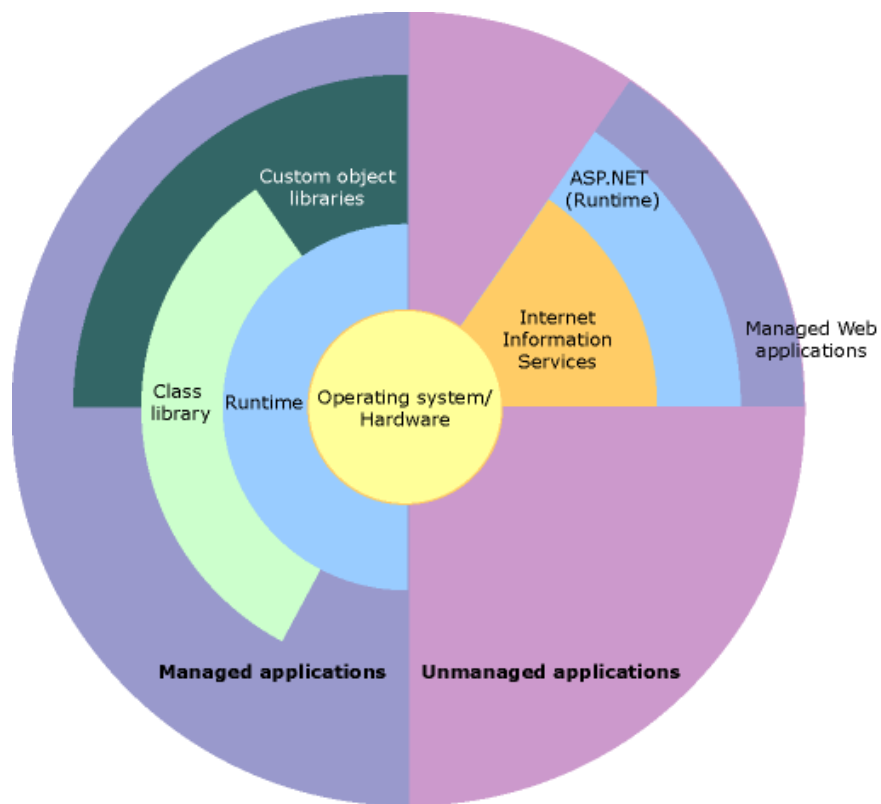
- To provide a consistent object-oriented programming environment, whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that guarantees safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

Microsoft Visual Studio.NET is a next-generation development tool that expedites the development process and facilitates a simpler development process for .NET applications. Although it is possible to build any type of .NET application, including Building Blocks, using a simple text editor, Visual Studio .NET is the recommended tool for building Blackboard Building Blocks.

.NET Framework

The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework. The runtime can be compared to an agent that manages code at execution time, providing core services such as memory management and thread management, while also enforcing strict type safety and other forms of code accuracy that ensure security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that may be used to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

The following image represents .NET in context:



Multiple Language Support

The .NET Framework Common Language Runtime makes it possible to develop applications that can target the .NET subsystem from a variety of programming languages. Most examples in this manual are in C#. The following are a subset of the languages supported by .NET:

- APL
- C#
- C++
- COBOL
- Component Pascal
- Eiffel

- Fortran
- Haskell
- J#
- JavaScript
- Mercury
- Mondrian
- Oberon
- Pascal
- Python
- Scheme
- SmallTalk
- Standard ML
- Visual Basic

Examples

Below are several examples of building Blackboard System Extensions using various languages:

The following is a Visual Basic example:

```
Imports Blackboard.Data.Course
Imports Blackboard.Persist

Public Class WebForm1

    Private Sub Page_Load()
        ' Load up the list of courses that the current user is
        ' enrolled in.
        Dim container As BbDatabaseContainer
        Dim courses As Course()
        Dim courseInstance As Course
        Dim item As ListItem

        container = BbDatabaseContainer.DefaultInstance
        courses = Course.LoadEnabledByUserId(container, New BbId(container,
            GetType(Blackboard.Data.User.User), 1))
        For Each courseInstance In courses
            item = New ListItem(courseInstance.CourseId,
                courseInstance.Id.Key.ToString)
            blackboardCourseDropDownList.Items.Add(item)
        Next

    End Sub

End Class
```

The following is a C# example:

```
using System.Web.UI.WebControls;

using Blackboard.Data.Course;
using Blackboard.Persist;

namespace BBCSSample
{
    public class WebForm1 : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.DropDownList blackboardCourseDropDownList;

        private void Page_Load()
        {
        }
    }
}
```

```
{
    BbDatabaseContainer    container;
    Course []             courses;

    container = BbDatabaseContainer.DefaultInstance;
    courses = Course.LoadEnabledByUserId (container,
        new BbId (container, typeof
            (Blackboard.Data.User.User), 1));
    foreach (Course course in courses)
    {
        ListItem item = new ListItem (course.CourseId,
            course.Id.Key.ToString ());
        blackboardCourseDropDownList.Items.Add (item);
    }
}
}
```

Building Block Overview

Overview

A .NET Building Block is a .zip or .war file that has a certain structure defined by Blackboard. This structure primarily consists of a file-directory hierarchy containing ASP.NET code tied together by a *Blackboard Learning System* manifest. There are many different types of Building Blocks that can be built, including new content types, collaboration tools and portal modules. This section discusses the definition of a .NET 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?](#)
 - [Things to do with a Building Block](#)
 - [Architecture Examples](#)
-

What is a Building Block?

Overview

A .NET Building Block is simply a set of files installed on the Blackboard Web 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 the *Blackboard Learning System* 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 .NET is simply an ASP.NET Web application, with some supplemental information provided for *Blackboard Learning System* to locate the resources within a Building Block. The .NET Software Development Kit, defines .NET Web applications and Web Services.

Entry points = Hyperlinks

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

.NET libraries

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

Access control

Access control is centralized in Blackboard.NET framework. Given a user ID (and a course ID, in the case of course related entitlements), a user can check for certain entitlements in `Blackboard.Security.Authorization.IsUserAuthorized()` (two overload methods one with and one without course ID argument).

Dynamic deployment

Building Blocks on the Microsoft .NET Platform may be installed and enabled without restarting the IIS server. This provides more flexibility in administrative options.

Things to Do With a Building Block

Overview

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

While entry points limit where Building Blocks appear in the *Blackboard Learning System* user interface, the possible behaviors of a Building Block are not as limited. Building Blocks may 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 the *Blackboard Learning System* 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 as course-specific hooks into a library reservation, can also be created.
- **Communication Tool.** Communication tools can be developed. For example, a hook to a different chat server.
- **Publish Web Services.** Publish certain data through Web services.
- **Portal Module Type.**

Developers can also build combinations of any of the above. An example might be a Building Block that bridges the *Blackboard Learning System* 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

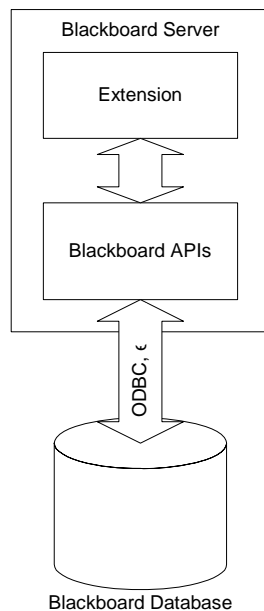
Overview

In order to illustrate what can be done with a Building Block, two basic architectures for extensions 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 type of Building Block.

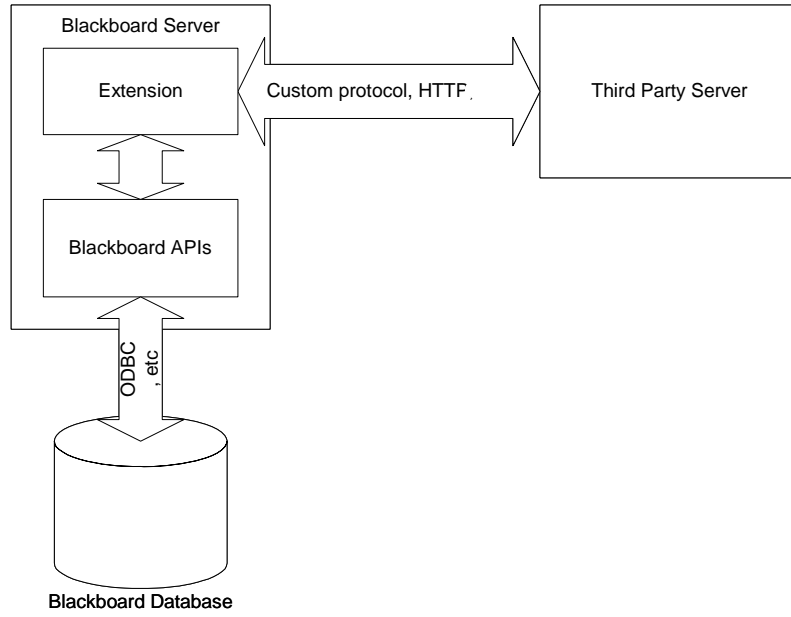
Below is an example of the plug-in architecture.



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 extension 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

Overview

The *Blackboard Learning System* exposes several .NET 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, Gradebook, and Authorization.

In this section

The topics in this section include:

- [Data Objects and Persistence APIs](#)
 - [Session, Portal and Authorization APIs](#)
-

Data Objects and Persistence

Overview

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.

Data Objects

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

The .NET data objects are currently tied to MS SQL server database containers for storing/retrieving data.

All data objects are in the system namespace `Blackboard.Data`.

The following are examples of available data objects:

- Announcement
 - Calendar
 - Content
 - Course (this includes CourseMembership and GroupMembership)
 - User
 - Gradebook (this includes LineItem and Score)
-

Persistence

Data persistence is achieved through the data objects themselves, and the `Blackboard.Persist` namespace. Each data class contains static `Save()` and `DeleteById()` methods. The `Save()` method provides intelligent insert/update operations based on the ID field of the persisted data object. In most cases, using the granular (by Id) load methods and the `Save()/DeleteById()` methods are sufficient. There are also various collection loaders that return lists of objects based on specific criteria. For example, `Content.LoadContentChildren()`.

The persistence methods automatically deal with database interaction, including opening and releasing connections, formatting SQL, and processing transactions.

The following example shows the steps required to create and save a Content object:

```
Content content = new Content();
content.ContentHandler = HANDLER_STRING;
content.ParentId = folder.Id;
content.IsAvailable = true;
content.IsDescribed = false;
content.IsFolder = false;
content.TextFormat = TextFormat.Plain;
content.Title = textTitle.Text;
content.Body = textBody.Text;
Content.Save( dbc, content );
```

Note that the example uses `Content.Save()`. `Save()` is actually defined on `BbObject`; `BbObject.Save()` could be used as well. The `dbc` argument is the

BbDatabaseContainer, which selects the proper database to use based on HTTP host headers. Call `BbDatabaseContainer.GetCurrentContainer()` to get a reference.

Session, Portal and Authorization APIs

Overview

The following topic describes the APIs for the following:

- Session
 - Portal
 - Authorization
 - Authentication.
-

Session

All requests to the application are associated with a session. This object retains information about the user and their authentication status. The ASP.NET `HttpContext.Current` is used throughout the .NET APIs for identifying the current Database container.

See the namespace `Blackboard.Platform.Session`

Portal

When a developer builds a .NET Building Block Portal Module, he or she sometimes needs to store and retrieve a module and/or user specific data (`customData`). A simple Portal API is provided to allow such operation.

See the namespace `Blackboard.Portal`

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 `Authorization` object. Building Blocks are required to include in the manifest a brief security descriptor that identifies permissions required to operate.

See the namespace `Blackboard.Security.Authorization`

Authentication

Authentication is handled automatically by the login subsystem of *Blackboard Learning System*. If a Building Block must ensure that authentication has occurred, it may call `Session.IsAuthenticated()`. If this method returns false, a login sequence may be initiated.

See the namespace `Blackboard.Platform.Session`

Using the Building Blocks APIs and Runtime

Overview

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

In this section

The following topics are included in this section:

- [Top-level Package Structure](#)
 - [The Blackboard Data Model](#)
 - [Strongly-typed Enumerations](#)
 - [Persistence Services](#)
-

Top-Level Package Structure

Overview

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

Top-level Packages

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

Namespace	Description
Blackboard.Data	Data package. This package, together with its sub-namespaces Announcement, Calendar, Task, and User, contains the definition of the <i>Blackboard.NET Learning System</i> core data model.
Blackboard.Persist	Persistence framework package. This package contains the core elements in the Building Blocks APIs and Runtime. Its sub-namespaces manage the persistence for each data object. These interfaces make up the core of the persistence framework.
Blackboard.Base	Package for miscellaneous classes that define core-level functionality or concepts.
Blackboard.Utils	Utility package. Contains application configuration and some tools
Blackboard.Platform	Contains Session class.
Blackboard.Portal	Contains Portal class
Blackboard.Security	Contains the Authorization, plug-in permission and permission classes

Blackboard Data Model

Overview

The data classes in the various sub-packages of `blackboard.data` provide almost complete coverage of the data that is tracked by the *Blackboard Learning System*.

Data sub-namespaces

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

Data sub-package	Classes contained
Announcement	Contains Announcements in the system and courses.
Calendar	Contains the <code>CalendarEntry</code> class that is used by the Calendar subsystem.
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 <code>ButtonStyle</code> class.
User	Contains core classes pertaining to users and other user information, such as an Address Book entry.
Gradebook	Contains Gradebook lineitem and score classes.
Navigation	Contains classes for creating and manipulation the course table of contents.

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 objects are limited to contain only a set of attributes and getter and setter methods for those attributes.

Strongly-Typed Enumerations

Overview

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 used throughout the .NET APIs to provide strong types for the runtime and development. They are declared in their corresponding name spaces. For example, name space `Course` contains an enumeration called `CoursePace`. Given this enumeration, the property `course.CoursePace` can be set as `CoursePace.InstructorLed`.

```
public enum CoursePace
{
    Unset,
    InstructorLed,
    SelfPaced
}
```

Tips and tricks

The first elements in some of the enumerations are defined as "Unset" for handling null values (in the database).

Blackboard Look and Feel

Overview

When creating a .NET Building Block it is useful to the end user to make the Building Block's interface fit seamlessly into the *Blackboard Learning System*. There are two ways this may be accomplished: the Blackboard Web Controls, 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 Web Controls](#)
 - [Icons](#)
-

Building Blocks Web Controls

Overview

The Blackboard for .NET Web Controls is a set of .Net custom server controls, which provide a common Blackboard look and feel for Building Block developers.

While it is not required to use the Blackboard for .NET Web Controls, 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 Learning System UI* becomes available in different contexts, such as Wireless Application Protocol (WAP) or text-only versions for high accessibility.

Note: Additional information regarding server controls can be found in Microsoft.NET references.

Controls

The following controls are included with the application. In many of the samples, the XML namespace prefix "cc1" is used in the tags. The prefix is defined in the "Register" directive in each .aspx file.

```
<%@ Register TagPrefix="cc1" Namespace="Blackboard.Web.BlackboardWebControls"
  Assembly="BbWebControls, Version=1.0.0.0, Culture=neutral,
  PublicKeyToken=7f9389376ce2c01e" %>
```

Note: All Blackboard for .NET Web Controls inherit all of the properties (Attributes) from WebControl. The Attributes defined below are Blackboard Web Controls' additional properties. There are additional controls in the BbWebControls.dll assembly than these composite controls listed below. Those are base controls used to build composite controls and are not recommended for use by Building Block developers.

BbActionBarControl

Definition:

Acts as a container control to hold BbActionGroupControls

Attributes:

Summary – summary of the current action bar

BbActionGroupControl

Definition:

Acts as a container control to hold BbActionItems

Attributes:

Summary – Summary for the current ActionGroup

Action – current action

Icon – icon name to use for the ActionGroup, use "/images/..."

IconAlt – Alt text for the Icon

Items – list of ActionItemControl.

MaxItems – Maximum number of ActionItems included in this group that should be displayed in the action bar. These are displayed in a horizontal format. Additional items included in the group that exceed the MaxItems value are added to a dropdown selection box in the action bar.

BbActionItemControl

Definition:

List of action items defined in a BbActionGroup.

Attributes:

Href – target URL

Icon – Icon used for this ActionItem

Target – target browser frame/window name

Title – text description of the ActionItem

BbBodyControl

Definition:

Default Blackboard ASP.Net body tag. This includes the default Blackboard style sheet. To ensure the Building Block page has the same look and feel as the rest of the system when BbWebControls are used, use this control in all ASP.NET pages.

BbBreadcrumbBarControl

Definition:

Provide navigation traces. The current page uri, current page referrer and the Label are registered. Combined with the Breadcrumb information from the referrer, a series of traces can be reconstructed.

Attributes:

Label – Text associated with breadcrumb item

BbButtonControl

Definition: Hides the details of a button object

Attributes:

ImageName – then file name of the image

ButtonMode – On or Off

URL – the target URL.

ButtonImage – the name of the button. Can be specified using the enumeration

Blackboard.Web.BlackboardWebControls.BbButtonControl.ButtonName

Alt - Alt tag in the code.

BbCalendarControl

Definition:

Provides a convenient way to select dates in a web form.

Attributes:

AutoPostBack – If true, a postback occurs on every interaction with the control and the server controls are used to update the page rendering.

MaxDate – Get the maximum allowable date

MinDate – Get the minimum allowable date

SelectedDate – The date selected on calendar, defaults to current date.

SelectedDay – The day selected on calendar

SelectedHour – The hour selected on calendar

SelectedMinute – The minute displayed on calendar

SelectedMonth – The month selected on calendar

SelectedYear – The year selected on calendar

ShowTime- If true, time is included in calendar display

Examples:

This following sets the calendar to May 15, 1964 1:35 PM

```
<cc1:BbCalendarControl id="Bbcalendar1" runat="server" ShowTime="true"
SelectedMonth=5 SelectedDay=15 SelectedYear=1964 SelectedHour=13
SelectedMinute=35>
</cc1:BbCalendarControl>.
```

BbColorPickerControl*Definition:*

Used to pick colors in a Web form from an existing color palette. It utilizes the standard Blackboard color picker javascript.

Attributes:

Alt – Alt text for the control
 Border – border width of the color picker
 Height – height of the color picker (pixels)
 Width – width of the color picker (pixels)
 HiddenName – name of the hidden field for the color field
 ImageName – name of the image name of the color picker
 Src – the default icon for the color picker

BbHtmlInput*Control:

BbHtmlInputButtonControl
BbHtmlInputCheckBoxControl
BbHtmlInputHiddenControl
BbHtmlInputImageControl
BbHtmlInputRadioButtonControl
BbHtmlInputTextControl

Definition:

In ASP.NET, to avoid name conflicts, the resulting input tags rendered from `<asp:Input*/>` controls has an automatically generated "name" attribute, which is not available to Building Block developers. Blackboard client validation requires the name to be passed as a parameter. BbHtmlInput*Controls provides a way to override the Name property of these controls.

Attribute:

Name – the end "name" attribute of the input

BbOKButton*Definition:*

OK button to initiate an action.

Attributes:

Icon – the Icon to use for the button. Default to Blackboard "OK" icon
 Target – the target window/frame name
 URL – the target URL

BbPluginLinkControl*Definition:*

For module view page only, to replace the usual `<a/>` tag for relative links only. On the module view page, only a full URL or an absolute path `"/"` may be used. If a Web application uses relative paths in the view page, then BbPluginLinkControl should be used.

Attributes:

Pref – the relative path of the link

Examples:

```
<ccl:BbPluginLinkControl id="Bblink1" PRef="BbTestWC.aspx" runat="server">Test Web Controls</ccl:BbPluginLinkControl><BR>
```

BbReceiptControl*Definition:*

Standard receipt page used after certain action.

Attributes:

Content – content

OKURL – the target url for the ok button
Subsubtitle – Inner text to page subtitle
Subtitle – Text on page subtitle
Title - Text added to title bar.

BbSpacerControl

Definition:

Spacer adds spacing on the page

Attributes:

Border - border width of the spacer
HSpace - horizontal space
VSpace – vertical space
SHeight – Spacer height
Src – the background image path

BbStepContentHiddenControl

Definition:

A type of BbStepControlItemControl. Hidden field used in a step content

Attributes:

Name – name of the hidden field
Value – value of the hidden field

BbStepControlItemControl

Definition:

Item under step control.

Attributes:

Label – label for the content
IsLabelBold – make the label bold
IsRequired – denote the item required
IsWrap - If IsWrap=true, the label and the content will be on different lines. If false, they will be on the same line.

BbStepControl

Definition: Step control. The BbStepControlItemControls should be nested within this control

Attributes:

Step – the step number
Title – the title text
BeginVSpace - the space before the title text

BbStepLabelComboControl

Definition:

A type of BbStepControlItemControl. It provides a combo (label and value pair)

Attributes: Inherits BbStepControlItemControl attributes.

Text – the label

Body:

Value of the label

Examples:

```
<cc1:BbStepLabelComboControl id="BbSLC1" runat="server" Label="This is a Label Combo:"  
IsLabelBold="false">Value</cc1:BbStepLabelComboControl>
```

BbStepInputComboControl*Definition:*

A type of BbStepControlItemControl. It provides a combo (label and input)

Attributes: Inherits BbStepControlItemControl attributes.

Body:

Any input <asp:Input*/> or html <input /.> tags

Examples:

```
<cc1:BbStepInputComboControl id="BbSIC1" runat="server" Label="This is an Input Combo  
- Input" IsLabelBold="false">  
  <input name="Name" value=""></input>  
</cc1:BbStepInputComboControl>
```

BbStepSubmitControl*Definition:*

A Special step control for submit step.

Attributes:

CancelURL – the target URL for cancel action

Step – the step number

SubmitURL – the target URL for submit

BbTitleBarControl*Definition:*

Title bar page

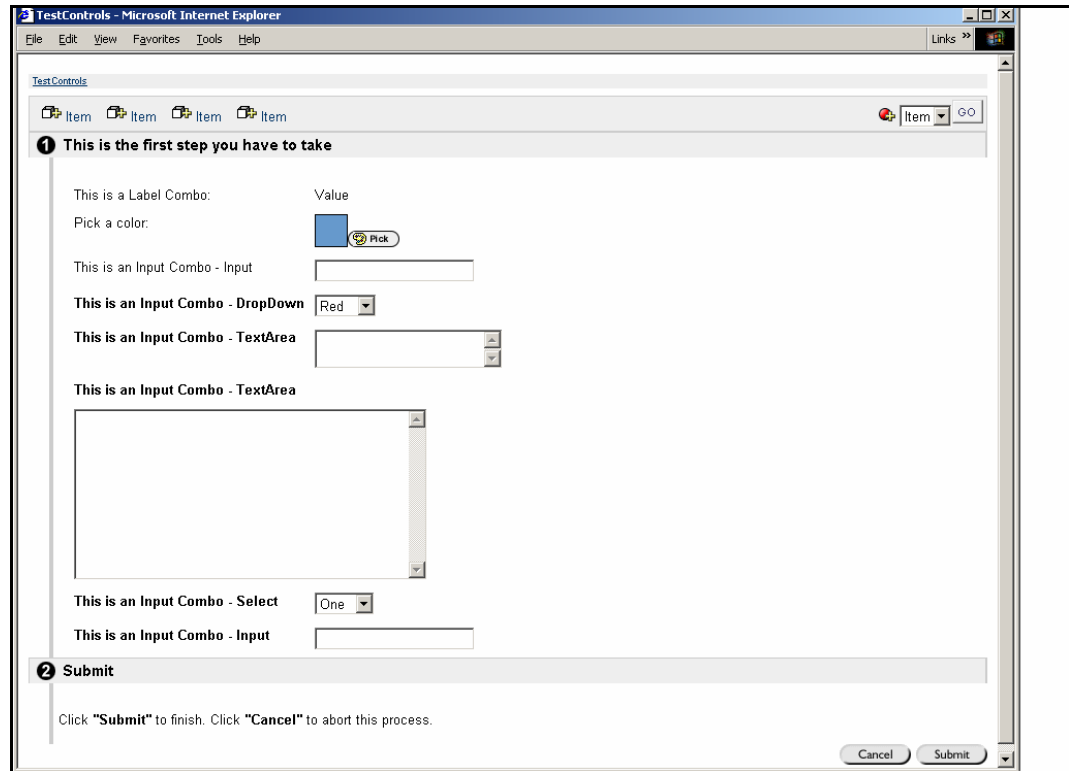
Attributes:

Icon – the icon path to use for the title bar

Title – title text

Example

The following is a sample page using Blackboard Web controls.



The HTML for the example above is generated by the following ASP.NET code:

```
<% Register TagPrefix="cc1" Namespace="Blackboard.Web.BlackboardWebControls"
Assembly="BbWebControls,
Version=1.0.0.0,Culture=neutral,PublicKeyToken=7f9389376ce2c01e" %>
<% Page language="c#" Src="TestControls.aspx.cs" AutoEventWireup="false"
Inherits="BbAPITestTool.TestControls" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>TestControls</title>
    <meta content="Microsoft Visual Studio 7.0" name="GENERATOR">
    <meta content="C#" name="CODE_LANGUAGE">
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
    <SCRIPT type='text/javascript' src="/javascript/validateForm.js"></SCRIPT>
    <SCRIPT type='text/javascript'>function validate_form(form) {
      return validateForm();}
    </SCRIPT>
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="TestControls" method="post" runat="server" onsubmit="return
validate_form(this)">
      <cc1:bbbbodycontrol id="BbBodyControl1" runat="server">
        <cc1:BbBreadcrumbBarControl id="BbBreadcrumbBarControl1"
runat="server" Label="TestControls"></cc1:BbBreadcrumbBarControl>
        <cc1:BbActionBarControl id="BbActionBarControl1" runat="server">
          <cc1:BbActionGroupControl id="BbActionGroupControl1"
runat="server" MaxItems="4">
```

```

        <ccl:BbActionItemControl id="BbActionItemControl1"
runat="server"></ccl:BbActionItemControl>
        <ccl:BbActionItemControl id="Bbactionitemcontrol2"
runat="server"></ccl:BbActionItemControl>
        <ccl:BbActionItemControl id="Bbactionitemcontrol3"
runat="server"></ccl:BbActionItemControl>
        <ccl:BbActionItemControl id="Bbactionitemcontrol4"
runat="server"></ccl:BbActionItemControl>
        <ccl:BbActionItemControl id="Bbactionitemcontrol5"
runat="server"></ccl:BbActionItemControl>
        <ccl:BbActionItemControl id="Bbactionitemcontrol6"
runat="server"></ccl:BbActionItemControl>
        <ccl:BbActionItemControl id="Bbactionitemcontrol7"
runat="server"></ccl:BbActionItemControl>
    </ccl:BbActionGroupControl>
</ccl:BbActionBarControl>
    <ccl:BbStepControl id="BbStepControl1" title="This is the first step
you have to take" runat="server" Step="1">
    <ccl:BbStepLabelComboControl id="BbStepLabelComboControl1"
runat="server" Label="This is a Label Combo:"
IsLabelBold="false">Value</ccl:BbStepLabelComboControl>
    <ccl:BbColorPickerControl id="Bbcolorpickercontrol2"
runat="server" Label="Pick a color:" IsLabelBold="false" ImageName="bg_color"
HiddenName="bgColor"></ccl:BbColorPickerControl>
    <ccl:BbStepInputComboControl id="BbStepInputComboControl1"
runat="server" Label="This is an Input Combo - Input" IsLabelBold="false">
    <input name="Name" value=""></input>
    </ccl:BbStepInputComboControl>
    <script type="text/javascript">
        formCheckList.addElement(new
inputText({maxlength:100,invalid_chars:/[<>]/g,ref_label:"First
Name",name:"Name",minlength:1}));
    </script>
    <ccl:BbStepInputComboControl id="Bbstepinputcombocontrol2"
runat="server" Label="This is an Input Combo - DropDown">
    <asp:DropDownList id="DropDownList1"
runat="server"></asp:DropDownList>
    </ccl:BbStepInputComboControl>
    <ccl:BbStepInputComboControl id="Bbstepinputcombocontrol3"
runat="server" Label="This is an Input Combo - TextArea">
    <TEXTAREA rows="2" cols="20"></TEXTAREA>
    </ccl:BbStepInputComboControl>
    <ccl:BbStepInputComboControl id="Bbstepinputcombocontrol6"
runat="server" Label="This is an Input Combo - TextArea" IsWrap="true">
    <TEXTAREA rows="10" cols="40"></TEXTAREA>
    </ccl:BbStepInputComboControl>
    <ccl:BbStepInputComboControl id="Bbstepinputcombocontrol4"
runat="server" Label="This is an Input Combo - Select">
    <SELECT name="t_select">
        <OPTION value="1" selected>One</OPTION>
        <OPTION value="2">Two</OPTION>
        <OPTION value="3">Three</OPTION>
        <OPTION value="4">Four</OPTION>
        <OPTION value="5">Five</OPTION>
    </SELECT>
    </ccl:BbStepInputComboControl>
    <ccl:BbStepInputComboControl id="Bbstepinputcombocontrol5"
runat="server" Label="This is an Input Combo - Input">
    <asp:TextBox id="Textbox5" runat="server"></asp:TextBox>
    </ccl:BbStepInputComboControl>
</ccl:BbStepControl>
    <ccl:BbStepSubmitControl id="BbStepSubmitControl1" Step="2"
runat="server"></ccl:BbStepSubmitControl>

```

```
        </cc1:bbbodycontrol>  
    </form>  
</body>  
</HTML>
```

Icons

Overview

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

Overview

Content Building Blocks 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](#)
 - [Using Course Documents](#)
 - [Context Passing](#)
 - [Interacting with the Gradebook](#)
 - [Using the File System](#)
-

Entry Points

Overview

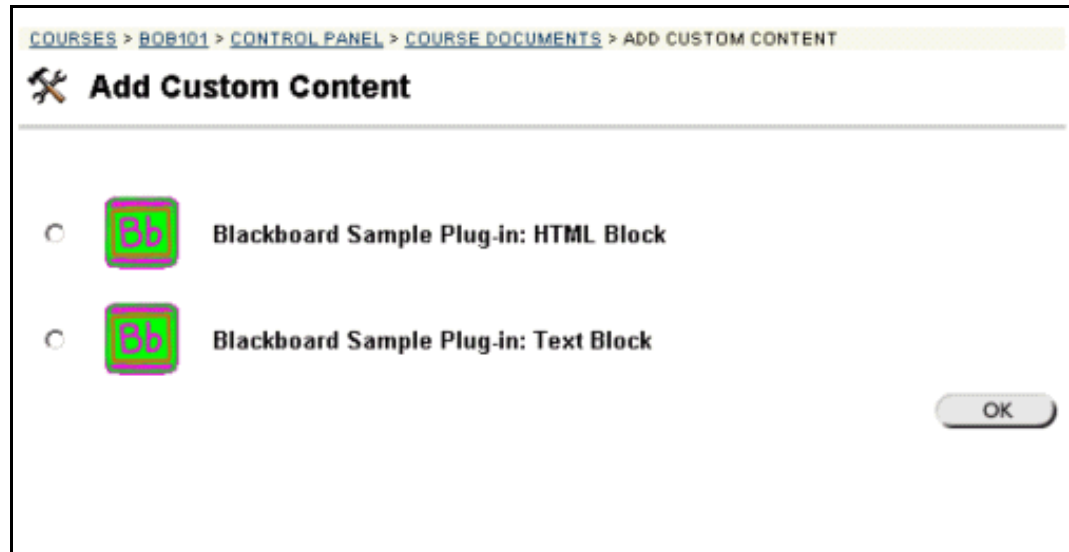
Entry points are defined for each content handler included in the manifest. Content handlers are used to render a Building Block's links for creating custom content types. Information on the manifest can be found in the [Blackboard Learning System Manifest](#) topic of this document.

The scripts provided in the Content Handler definition are the entry points of a Building Block in a Content Area (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 example below demonstrates create action rendering.

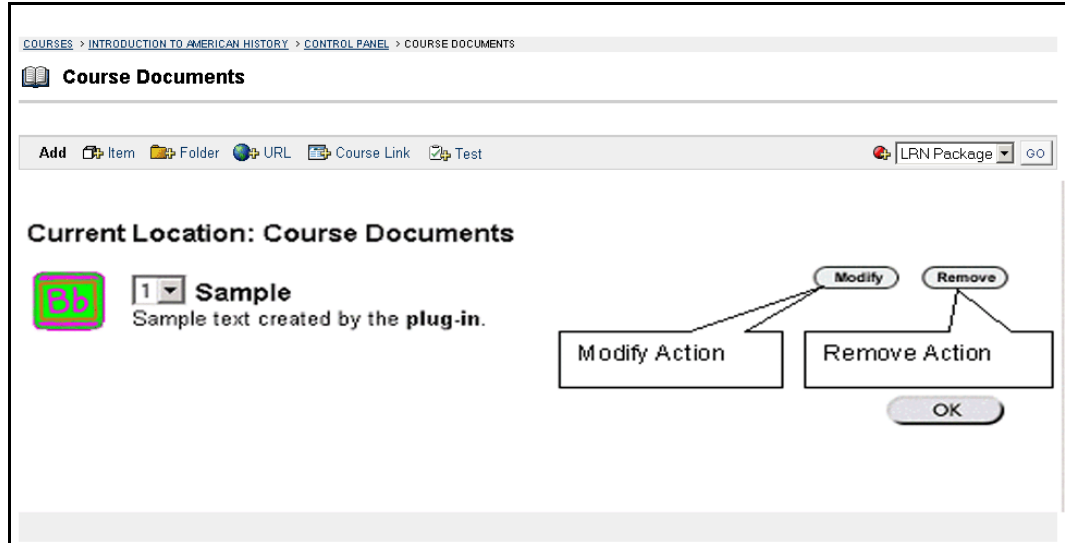


The create script must accept the following parameters:

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 Course Content

Overview

Course Documents are also referred to as Content Items. The key ability of a content extension 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 document object

Creating a document object is as simple as instantiating the data object and setting its attributes. The text that is entered via `CourseDocument.SetBody()` should be valid HTML, as that is what is used by the *Blackboard Learning System* core when rendering the document. The body attribute is actually manipulated by a `FormattedText` object that hides the details of *Blackboard Learning System* processing of Smart Text, Plain Text, and HTML. The following code is an example of a Course Document:

```
BbDatabaseContainer dbc = BbDatabaseContainer.GetCurrentContainer();

Content new_content = new Content();
new_content.CourseId = courseid;
new_content.Body = "This is a course content";
new_content.IsAvailable = true;
new_content.Title = "Course content title"
new_content.AllowGuests = true;
new_content.AllowObservers = true;

Content course_documents = Content.GetCourseDocumentsByCourseId(dbc, courseid);

new_content.ParentId = course_documents.Id;
Content.Save(m_current_dbc, new_content);
```

Using ID Objects

Look at the ID object obtained through a call to `Content.GetCourseDocumentsByCourseId()` in the preceding example. IDs encapsulate information, uniquely identifying an object. This includes any primary keys, the object type, and current database container. In an environment that has multiple Virtual Installations (VI), data with the same PK may exist in different Virtual Installations. The uniqueness is identified by database containers.

Note:

1. `BbId` object can be created with database container, data type and PK
 2. `ToString()` can be used to get a string representation of the ID object.
-

Save the document object

Once the object has been created it is saved by calling the static "Save" method of that class. If the ID in the persisted object already exists, the "Save" action results in an update.

Loading a Document

In addition to persist objects the developer works with existing data loaded from the database. Different load methods are provided for different types of data.

The following code demonstrates loading a document.

```
BbId contentId = new BbId( dbc, typeof(Content), Tools.getPKParam( "content_id" ) );  
Content content = (Content)BbObjectPersistence.LoadById( dbc, typeof(Content),  
contentId );
```

Context Passing

Overview

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

Using Context Passing APIs

The context passing APIs allow the *Blackboard Learning System* 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 the *Blackboard Learning System*. 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 Learning System* 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/contextWrapper` and include a URL template in the `url` parameter.

Below is an example of a deferred expansion URL:

```
/webapps/contextWrapper?encrypt=y&url=http://example.com/page?uid=@X@user.user_id@X@
```

The benefit of this approach is that there is an intermediate step to invoke the `contextWrapper`. 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 the *Blackboard Learning System*), can use to verify requests.

Variables

The following variables may be used in URL templates:

- `course.url`. The base URL for all files referenced in the course.

- `course.course_id`. The simple abbreviation for a course, for example, BIO101.
- `course.batch_uid`. The external identifier for a course.
- `user.user_id`. The login name for a user.
- `user.batch_uid`. The external identifier for a user.
- `content.url`. The base URL for files associated with a given piece of content. For example, if users are allowed to upload files into a document the referencing URL can be generated by the following template:
`@X@content.url@X@/uploaded_file`

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.

Interacting with the Gradebook

Overview

Content Building Blocks may need to interact with the Gradebook. This is done through the Gradebook API, which exists in the namespace `Blackboard.Data.Gradebook` and `Blackboard.Persist.BbObjectPersistence`

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. This is particularly useful if the extension is a bridge to an external assessment engine.

Writing Tool Building Blocks

Overview

Writing a Tool Building Block can be a very open-ended task which may 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 the *Blackboard Learning System*.

Applications

In the *Blackboard Learning System* 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 five 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 portal 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.

Communication Tool – A Communication Tool enables users to communicate with each other, for example, chat tools.

Course Tools – Course Tools extend the features of a course Web site.

General Development Tasks

Overview

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

Overview

Authentication is performed by the *Blackboard Learning System*, 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:

```
using Blackboard.Platform;

Session bbSession = Session.GetCurrentSession();
.
.
if (! bbSession.IsAuthenticated)
{
    // perform error processing
    return;
}
```

See `Blackboard.Platform.Session`

Authorizing Users

Overview

User authorization is performed through `Blackboard.Security.Authorization`.

Authorization example

This example demonstrates how to use the method `isUserAuthenticated()` for entitlement. The entitlement context is defined as static properties in the `Authentication` class and the permission types are defined in the `Blackboard.Security.Permission` as an enumeration.

The following is an authorization example:

```
//roles that are allowed access
Authentication.IsUserAuthorized(BbId userid, string AuthContext, Permission);

//roles that are allowed access to course related contents
Authentication.IsUserAuthorized(BbId userid, BbId courseid, string AuthContext,
Permission);
```

Building a Building Block

Overview

This section describes the areas outside of programming that developers should take into account when creating a Building Block.

Note: All of the code described in this section is included in the sample extension package delivered as part of the Building Blocks on the Microsoft .NET Platform Software Developer Kit.

In this section

The following topics are included in this section:

- [Development environment](#)
 - [Deciding what to build](#)
 - [Debugging the Extension](#)
-

Development Environment

Overview

Developers may create a Building Block with the Microsoft.Net SDK. The Building Block should contain ASP.NET files and related code behind classes.

Development Tools

Install the Visual Studio .NET integrated development environment. This will install the development tools as well as the Microsoft .NET Framework.

Assemblies

For some Building Blocks, the developer may include additional assemblies. Simply putting the user assemblies under the /bin directory of the Web application will make the DLLs available to the Building Block. Use file references to access the Blackboard DLLs. Adding `<add assembly="..." />` to the compilation element of the web configuration for each Blackboard DLL will dynamically link this assembly to these resources during compilation.

Example:

```
compilation defaultLanguage="c#" debug="true">
  <assemblies>
    <add assembly="BbPlatform, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=4dc438b31d4187f1, Custom=null"/>
    <add assembly="BbWebControls, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=7f9389376ce2c01e, Custom=null"/>
  </assemblies>
</compilation>
```

Deciding What to Build

Overview

The Building Block sample that is provided is a simple example 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

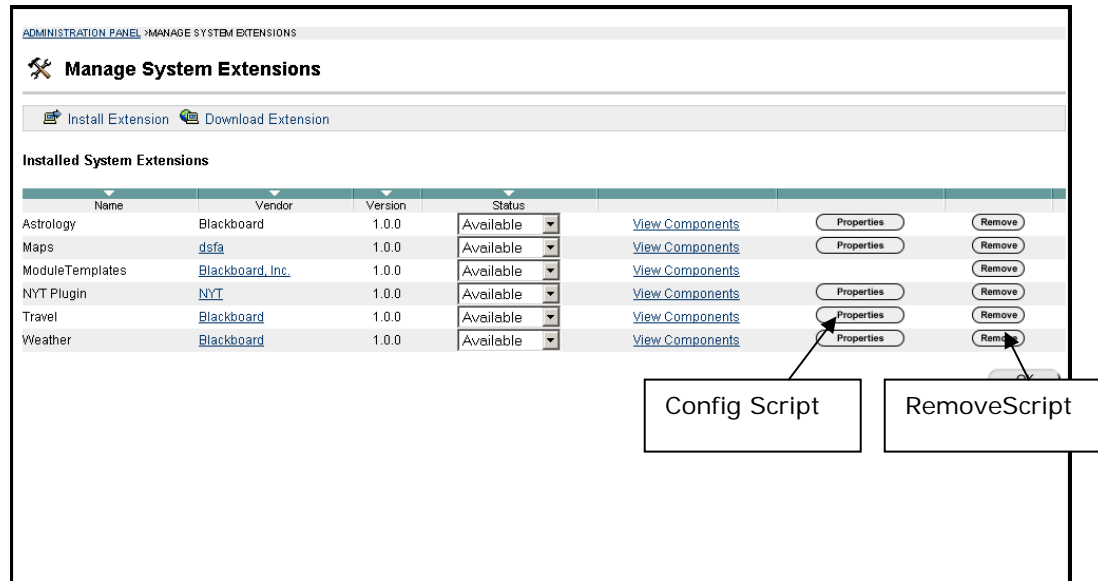
Links are different from content handlers and configuration scripts in that they are not clearly defined by http actions. Instead, they are defined with a link element and a type attribute. This is because in the tool-oriented areas of the application (course tools and communications) a single entry 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 the user 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 the property `Blackboard.Util.BbApplication.GetCurrentApplication().CurrentPlugin.HomeDirectory`.

The following example demonstrates rendering the configure and remove scripts.



ADMINISTRATION PANEL > MANAGE SYSTEM EXTENSIONS

Manage System Extensions

Install Extension Download Extension

Installed System Extensions

Name	Vendor	Version	Status			
Astrology	Blackboard	1.0.0	Available	View Components	Properties	Remove
Maps	dsfa	1.0.0	Available	View Components	Properties	Remove
ModuleTemplates	Blackboard, Inc.	1.0.0	Available	View Components		Remove
NYT Plugin	NYT	1.0.0	Available	View Components	Properties	Remove
Travel	Blackboard	1.0.0	Available	View Components	Properties	Remove
Weather	Blackboard	1.0.0	Available	View Components	Properties	Remove

Config Script

RemoveScript

Build a Remove Script

By providing a remove script, the developer can perform any necessary cleanup when the Building Block is removed from the *Blackboard Learning System*. 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 the *Blackboard Learning System*. 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.

Building Block XML Packaging Format

Overview

Packaging a Building Block includes creating a .ZIP file that includes additional data used by the *Blackboard Learning System* 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 Extension](#)
-

Web Archive Overview

Overview

A Web archive file is a .ZIP file conforming to the sub-directory layout described below.

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
bin/	.dll files included for the application. These may include libraries that are developed or third party libraries included as utilities.
WEB-INF/bb-manifest.xml	The Blackboard package definition (required by the <i>Blackboard Learning System</i> package specification).
/	The root of the Building Block. With the exception of WEB-INF, developers may organize this directory as they see fit. Typically, this will include the Building Block's aspx scripts, images, and so forth.

URLs

Overview

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 server-side transfers (for example, using `System.Web.Server.Transfer()`). Care must be taken in the construction of URLs as the evaluation of the URL is dependent on the method used.

Also, note that the base path of the Building Block will not be known until it is deployed on a system.

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, current plugin properties should be used.

`Blackboard.Util.BbApplication.GetCurrentApplication().CurrentPlugin` provides the current plugin context. `BbPlugin.Handle` can be used to construct the absolute path: `/webappsnet/handle/myasp.aspx`

Blackboard Learning System Manifest

Overview

The manifest is the set of directives the developer provides to *Blackboard Learning System* 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.

Action groups

There are several main action groups:

- **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 portal home page.
- **Module Types.** Custom logic used to display modules in the portal 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 Development](#) section of this manual.

Portal Modules, Module Types, and RSS Channels can also be defined in the manifest. The plugin must define the following, top-level elements.

Element	Description
Name	A name for the plugin.
Handle	A unique string value. In conjunction with the vendor id, it is meant to be unique within the system.
Description	A brief (up to 255 characters) description of the plugin.
Http-Actions	Defines the collection of entry points for editing content created by this handler.
Config	Script called for user-entered configuration options.
Remove	Not yet supported for the .NET plugins
Vendor	Information about the vendor or institution providing the plugin. Contains id, name, url and description.
Id	Four character identifier for the vendor
Name	Name for the vendor
URL	Link for the vendor home page.
Description	A brief (up to 255 characters) description for the plugin vendor.
Version	Version number for the plugin. String must be of the form n.n.n.
Request	Version information required for proper operation. Only one element is currently supported, bbversion.

Bbversion	The Blackboard version number required to install this extension. The minimum value for .NET extensions should be 6.0.14.
Webapp-type	Deployment type for this plugin. .NET plugins must use the value "Net" (not case sensitive).

```
<name value=".NET Sample Plugin"/>
<handle value="dotnet-sample"/>
<description value="This plugin a Blackboard.Net Sample Extension."/>
<version value="1.0.0"/>
<requires>
  <bbversion value="6.0.14"/>
</requires>
<vendor>
  <id value="bb"/>
  <name value="Blackboard for .NET Team"/>
  <url value="http://www.blackboard.com/" />
  <description value="Blackboard.Net" />
</vendor>
<http-actions>
  <config value="admin/Config.aspx"/>
  <remove value="admin/Remove.aspx"/>
</http-actions>

<!-- value required for proper deployment -->
<webapp-type value="Net" />
```

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.
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 <code>course_id</code> and <code>content_id</code> arguments. The <code>content_id</code> argument for creation references the parent folder.
Modify	The script called to modify the content created by this handler. The script must accept a <code>course_id</code> and <code>content_id</code> arguments. The <code>content_id</code> argument is the object to modify.
Remove	The script called to follow up on deletion of content. The script must accept a <code>content_id</code> argument. Note that the actual deletion is performed by the <i>Blackboard Learning System</i> . This is called to allow cleanup by the Building Block.
Icons	Collection of icons to display to the user.
Toolbar	Reserved.
List item	Not used. <i>Blackboard Learning System</i>

Example

The following example defines a simple content handler.

```
<content-handlers>
  <content-handler>
```

```

<name value=".NET Sample Content"/>
<handle value="resource/x-bb-dotnet"/>
<http-actions>
  <create value="ContentHandler.aspx"/>
  <modify value="ContentHandler.aspx"/>
</http-actions>
<icons>
  <toolbar value="images/bookopen_u.gif"/>
  <listitem value="images/bookopen_u.gif"/>
</icons>
</content-handler>
</content-handlers>

```

Defining an application

In *Blackboard Learning System* links should appear inside a `<link>` 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.
Name	The user-friendly name of the application.
Small-icon	Reserved.
Large-icon	Reserved.
Description	A description displayed to the user.
Links	The collection of links exposed by this application.
Link	A single link within the application.
Type	Type of the link. This must be one of: tool, communication, course_tool, user_tool, or system_tool.
Name	Name to display for the link.
URL	Relative path of the tool. This must be relative to the web application root (but not root-anchored).
Description	Description of the link to display.
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="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="system_tool"/>
        <name value=".NET Plugin Admin Panel Tool"/>
        <url value="AdminPanel.aspx" />
        <description value="Demonstrates adding tools to the system
administration tool." />
        <icons>
          <listitem value="images/tools_u.gif"/>
        </icons>
      </link>
    </links>
  </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 Specification Guide* 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. The exception is for mnemonic names for Blackboard-defined permissions or commonly requested Java permissions.

- **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 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=".NET Sample Plugin"/>
    <handle value="dotnet-sample"/>
    <description value="This plugin a Blackboard.Net Sample Extension."/>
    <version value="1.0.0"/>
    <requires>
      <bbversion value="6.0.14"/>
    </requires>
    <vendor>
      <id value="bb"/>
      <name value="Blackboard for .NET Team"/>
      <url value="http://www.blackboard.com/" />
      <description value="Blackboard.Net" />
    </vendor>
    <http-actions>
    </http-actions>

    <!-- value required for proper deployment -->
```

```

<webapp-type value="Net" />

<!-- Custom content types defined by this extension -->
<content-handlers>
  <content-handler>
    <name value=".NET Sample Content"/>
    <handle value="resource/x-bb-dotnet"/>
    <http-actions>
      <create value="ContentHandler.aspx"/>
      <modify value="ContentHandler.aspx"/>
    </http-actions>
    <icons>
      <toolbar value="images/bookopen_u.gif"/>
      <listitem value="images/bookopen_u.gif"/>
    </icons>
  </content-handler>
</content-handlers>

<links>
  <link>
    <type value="system_tool"/>
    <name value=".NET Plugin Admin Panel Tool"/>
    <url value="AdminPanel.aspx" />
    <description value="Demonstrates adding tools to the system administration
tool." />
    <icons>
      <listitem value="images/tools_u.gif"/>
    </icons>
  </link>
  <link>
    <type value="course_tool"/>
    <name value=".NET Plugin Control Panel Tool"/>
    <url value="ControlPanel.aspx" />
    <description value="Demonstrates adding tools to the course control
panel." />
    <icons>
      <listitem value="images/tools_u.gif"/>
    </icons>
  </link>
  <link>
    <type value="communication"/>
    <name value=".NET Communication Tool"/>
    <url value="CommunicationTool.aspx" />
    <description value="Demonstrates adding a tool to the course communication
menu." />
    <icons>
      <listitem value="images/tools_u.gif"/>
    </icons>
  </link>
  <link>
    <type value="tool"/>
    <name value=".NET Student Tool"/>
    <url value="CommunicationTool.aspx" />
    <description value="Demonstrates adding a tool to the course communication
menu." />
    <icons>
      <listitem value="images/tools_u.gif"/>
    </icons>
  </link>
</links>

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

```

```
<module-defs>
  <module-type ext-ref="dnsample" title="Sample.NET" uicreatable="true">
    <web-dir>/</web-dir>
    <web>
      <view>ModuleView.aspx</view>
      <edit>ModuleEdit.aspx</edit>
    </web>
  </module-type>
  <module type="dnsample" isadmin="false" useraddable="false" isdeletable="true"
isdetachable="true" title="DotNet Sample">
    <description>DotNet Sample</description>
    <ExtraInfo/>
    <module-groups>
      <module-group id="Everyone" />
    </module-groups>
  </module>
</module-defs>

  <permissions>
</permissions>
</plugin>
</manifest>
```

Packaging the Building Block

Overview

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

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.

Advanced Development Issues

Overview

Currently, Building Blocks are additions to the *Blackboard Learning System*. 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. Some of numerous possibilities for Building Blocks are discussed in this section.

Shipping additional libraries

As per the standard development practice in ASP.NET, Building Blocks may package custom and required libraries in the /bin directory of the Building Block.

Off-line tools

The term "off-line tools" refers to programs that are not run as Building Blocks, and are not run in the Blackboard ASP.NET process. The preferred technique is to do this via Web Services. You can use the Web Services Blackboard packaged with the Blackboard for .NET APIs or you can build your own (deploying them as a Building Block within Blackboard).

Debugging

Once a Building Block is installed, it is running in the IIS instance used by Blackboard. The easiest way to debug is to attach using either the Visual Studio.NET debugger, or by attaching via the CLR debugger that is shipped with the core .NET Framework. Note that the developer must attach to the aspnet_wp.exe process. Once they have attached, they can set breakpoints and pause execution as they normally would.

Troubleshooting

Introduction

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

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 .NET collections APIs.

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.config` file in the root directory.

Problem: Processing a form with user-entered markup results in an exception.

Solution: Set the page attribute `validateRequest` to `false`.

IIS Mapping Issues

Problem: Accessing any link for the Building Block results in either displaying the source code or a dialog asking to open the Asp.Net file.

Solution: Ensure that the IIS mappings for ASP.NET are correct. To fix IIS mappings for ASP.NET, follow these steps:

- Step 1** Run the `Aspnet_regiis.exe` utility:
- Click **Start**, then **Run**.
 - In the Open text box, enter `"cmd"`, and then select ENTER.
 - At the command prompt, enter the following and select ENTER:

```
"%windir%\Microsoft.NET\Framework\version\aspnet_regiis.exe" -i
```

In this path, `version` represents the version number of the .NET Framework that is installed on the server. This placeholder must be replaced with the actual version number when the command is entered.

- Step 2** Register the `Aspnet_isapi.dll`:
- Click **Start**, and then **Run**.
 - In the Open text box, enter the following, and then select ENTER:

```
regsvr32 %windir%\Microsoft.NET\Framework\version\aspnet_isapi.dll
```

Regsvr32 returns the results of the registration.

Module Building Blocks

Overview

.NET-based Building Blocks may install both Modules and Module types. A Module is a block of data that is rendered on a container page, such as the My Institution Tab. Each block is self-contained and is associated with a Module Type, which is the code that understands how to render and edit the contents of a Module. The Module Type code is defined by the <view>, <edit>, and <admin> attributes of the web element in the manifest.

Note: Module Building Blocks will only work if the *Blackboard Community System* is licensed.

Building Blocks may install Modules that reference existing Module Types on the system, or may install Modules that require a Module Type that is also included as part of the Building Block. See the *Blackboard Building Blocks Module Developer Guide* for more information on referencing existing Module Types.

Module Types

In most cases, custom Module Types are delivered as part of the Building Block, since extensions often require the display of proprietary data. At a minimum, the view is required. However, that will not allow any common portal entry points for user-entered data. An example is the Blackboard My Calendar module, which simply displays data that's aggregated from other tools in the system.

View is invoked when the Module is rendered in-line on the container page. View code must return an HTML fragment that does not include the following:

- <body> tags
- <html> tags
- <head> tags

Edit is invoked when the user selects the "edit" icon from the Module's control bar. The resulting page is rendered entirely within the content frame, thus giving more control over what elements can be rendered.

Additionally, an Admin attribute may be defined. This page allows Administrators to edit the global contents of a Module. By contrast, Edit is typically used to edit the personalized contents of a Module.

Module data (both global and personalized) is stored as XML in the database, and is accessed via the `Blackboard.Platform.CustomData` object, which allows setting custom attributes on the data.

Note: The View script is accessed via an HTTP proxy from the servlet engine; this is required to render the module content in-line with the other modules. As such, it is important to keep the service requests as short as possible for performance reasons.
