Blackboard Learn™ Content Management APIs & Xythos Architecture

John Barone, Senior Architect, Blackboard® Inc.

What will you learn in this session?

- An overview of what the Blackboard Content Management APIs are and how they relate to Blackboard Building Blocks™
- Moderately deep dive into Content Management API details
- What Xythos is and how it relates to Content Management within Blackboard Learn™
- What WebDAV is and why it's useful

Some Assumptions

- A high-level understanding of the Blackboard Building Blocks infrastructure, including:
  - Top-level package structure; e.g.:
    - blackboard.data, blackboard.persist, blackboard.base, etc.
  - Blackboard Learn Building Blocks data model; e.g.:
    - Objects such as announcements, course, user
  - Blackboard Learn Building Blocks persistence services; e.g.:
    - CourseDBLoader, CourseDBPersister, etc.
- Familiarity with the Blackboard .jsp tag library (<bbNG:>)
- Understanding of Building Blocks packaging/deployment, manifest, etc.

What are Building Blocks?

- Blackboard Building Blocks are web applications that developers build to extend the Blackboard Learn platform and to integrate Learn with external applications, content, or services.
- Building Blocks can include portal modules, content, specific tools, assessment engines, integration with library systems, campus bookstores, and much more.
- Building Blocks are packaged as web application module (i.e. .war files) and deployed either manually or automatically within Blackboard Learn.
- Availability of and access to individual Building Blocks can be controlled by Blackboard Learn administrators via system administration screens within the Learn UI.

Drivers for Developing Building Blocks

- Customer IT organizations may develop Building Blocks for:
  - Students (tools, communication, portal modules, portal tools, etc)
  - Instructors (control panel tools)
  - Administrator (admin control panel, headless tools/utilities)
  - No-one but themselves, just because they are “really cool” and they always wanted to do that…
- 3rd-party partners, open source developers (or even Blackboard Learn Product Development) may develop them to:
  - Provide cool new extensions not yet available within the Blackboard Learn platform.
  - Provide new features and enhancements “off cycle”; not strictly tied to the Blackboard Learn release schedule.

Where to Start?

- Think of use-cases first, before divining into code
  - What are the business workflows?
  - What UI screens are needed (story boards can be helpful)?
  - What sub-systems do you need to touch? What integration points are needed?
- Once the business and UI flows are fleshed-out, map the requirements to the APIs
  - Note that when the Blackboard Learn Content Management module is installed, all of the Learn APIs are available for use, ***not*** just those installed which expose specific Content Management UI hooks.

Other Resources

- Blackboard Learn API Javadoc
- Blackboard Content Management API Javadoc
- Blackboard Building Blocks Developers Guide
- Blackboard Building Blocks Tag Library Guide
- Blackboard Building Blocks: Getting Started Beginner/Intermediate Guide
- Blackboard Web Site
- Blackboard Open-Source Listserv (http://www.bb-opensource.org)
- Blackboard eduGarage (http://www.edugarage.com/display/BBDN/Documentation)
- Each other!

Blackboard Content Management API Overview

Blackboard Content Management APIs allow developers to:

- Programmatically create and manage content (files and folders) within the Blackboard Learn Content Collection; this includes:
  - Creating and accessing files and folders
  - Viewing and setting permissions
  - Viewing and setting various file and folder properties (e.g. folder quota)
  - Viewing and setting metadata
  - much more…
- Programmatically access several Content Management tools:
  - Learning Objects catalogue
  - Portfolios
  - Metadata Templates
  - Workflows

Blackboard Content Management API & Building Blocks

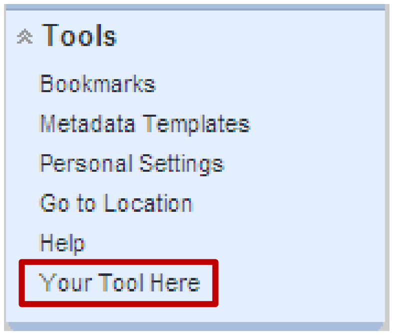
- The Content Management APIs are built on top of and are available from within the Build Blocks infrastructure
- Configure the Content Collection (UI) integration points via the bb-manifest.xml configuration file; e.g:

```
<links>
  <link>
    <type value="cs_tool"/>
    <name value="“Your Tool Here"/>
    <url value="tool.jsp" />
    <description value="“A most excellent Building Block” />
  </link>
  <link>
    <type value="cs_modify_file"/>
    <name value="“Your Tool Here"/>
    <url value="modifyfile.jsp" />
    <description value="“You can place Building Blocks on the modify pages!" />
  </link>
</links>
```

Content Collection Integration Points (i.e. UI Hooks)

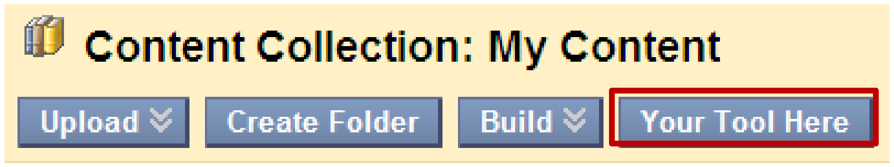
- Content Collection Tools

  Add links to the Tools box in Content Collection folder and shortcut views.



- Content Collection Action Bar

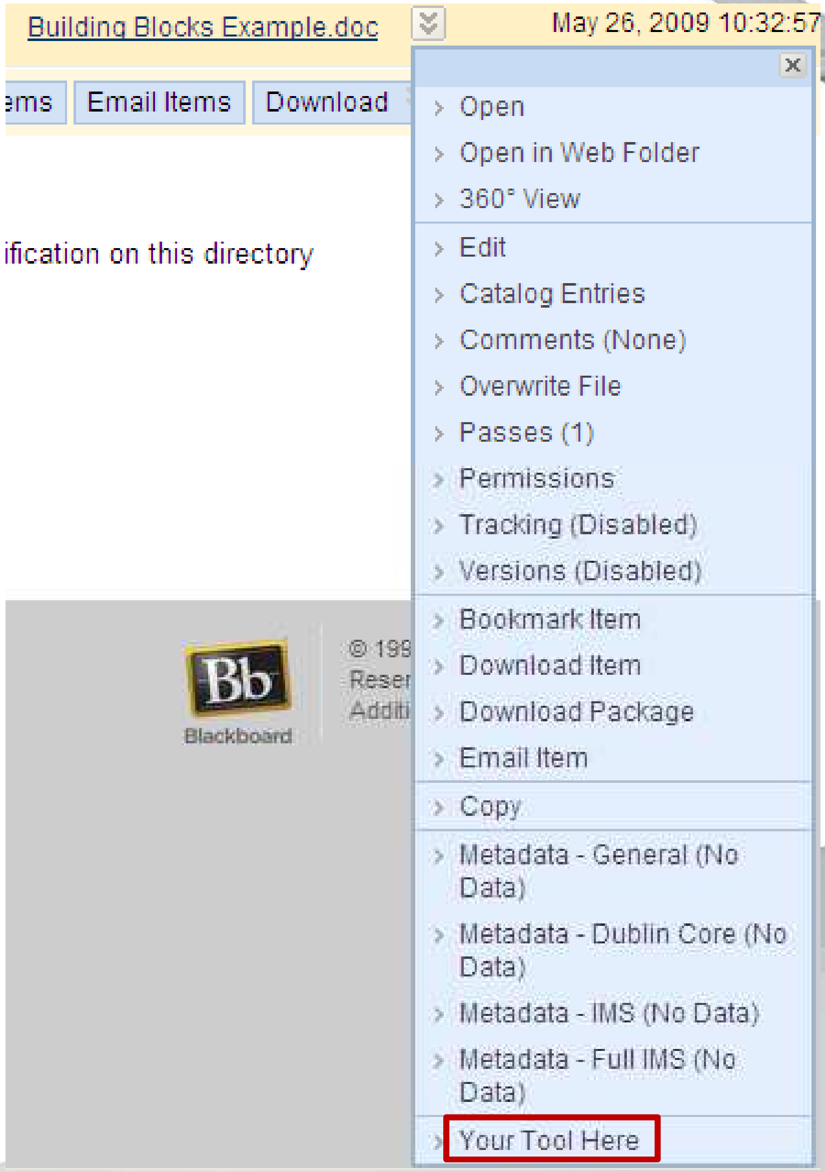
  Add a button link to the Action Bar on the Content Collection folder listing pages. Action Bar Building Blocks act on the files and folders currently selected in the folder listing.



Content Collection Integration Points (cont.)

- File/Folder contextual menu

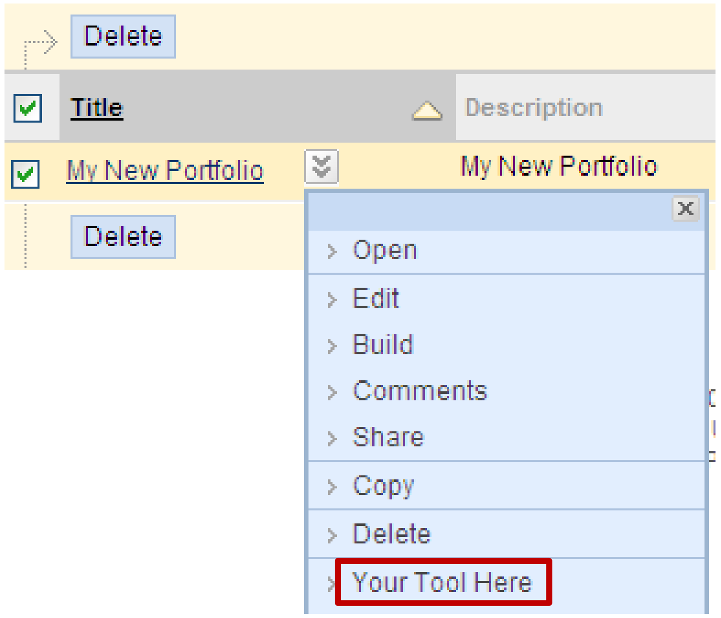
  Add a link within the contextual menu for files and folders within the Content Collection



Content Collection Integration Points (cont.)

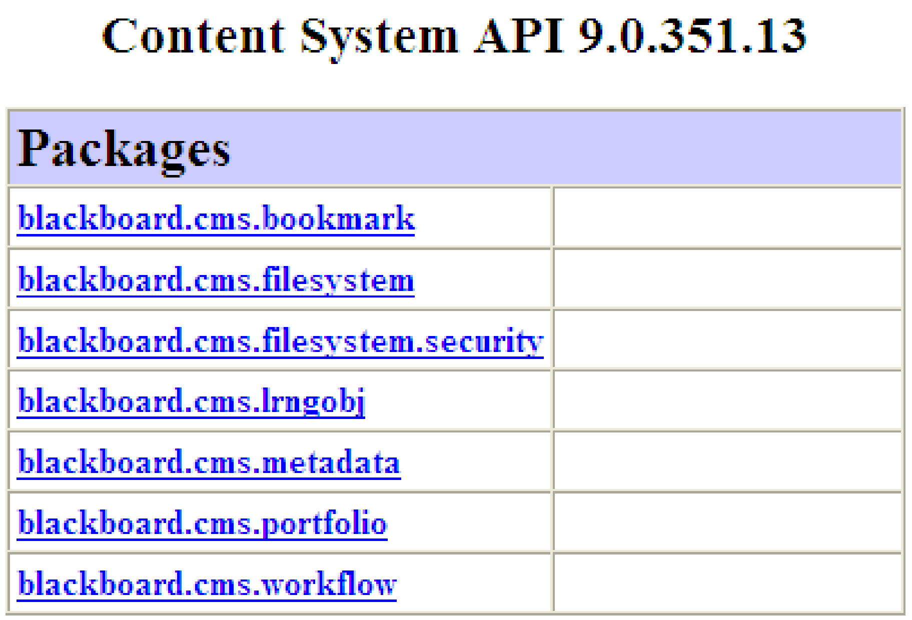
- Manage Portfolios/My Portfolios

  Add a link within the contextual menu for Portfolios within the Content Collection



Content Management API Data Object Packages

- Data object definitions reside in the various sub-packages of *blackboard.cms.**
- The sub-packages are grouped into functional areas such as `filesystem` or `portfolio`



Content System API 9.0.351.13

Packages	
blackboard.cms.bookmark	
blackboard.cms.filesystem	
blackboard.cms.filesystem.security	
blackboard.cms.lrngobj	
blackboard.cms.metadata	
blackboard.cms.portfolio	
blackboard.cms.workflow	

Data Object Sub-Packages

`bookmark`	Contains objects associated with Bookmarks in the system
`filesystem`	Contains objects used by the filesystem subsystem. This is one of the primary sub-packages within the Blackboard Management APIs, and contains the objects used to create and manage files and folders, as well as the “CSContext” object which is part of the Blackboard Content System permissions framework

DEV CON '09 @BbWorld

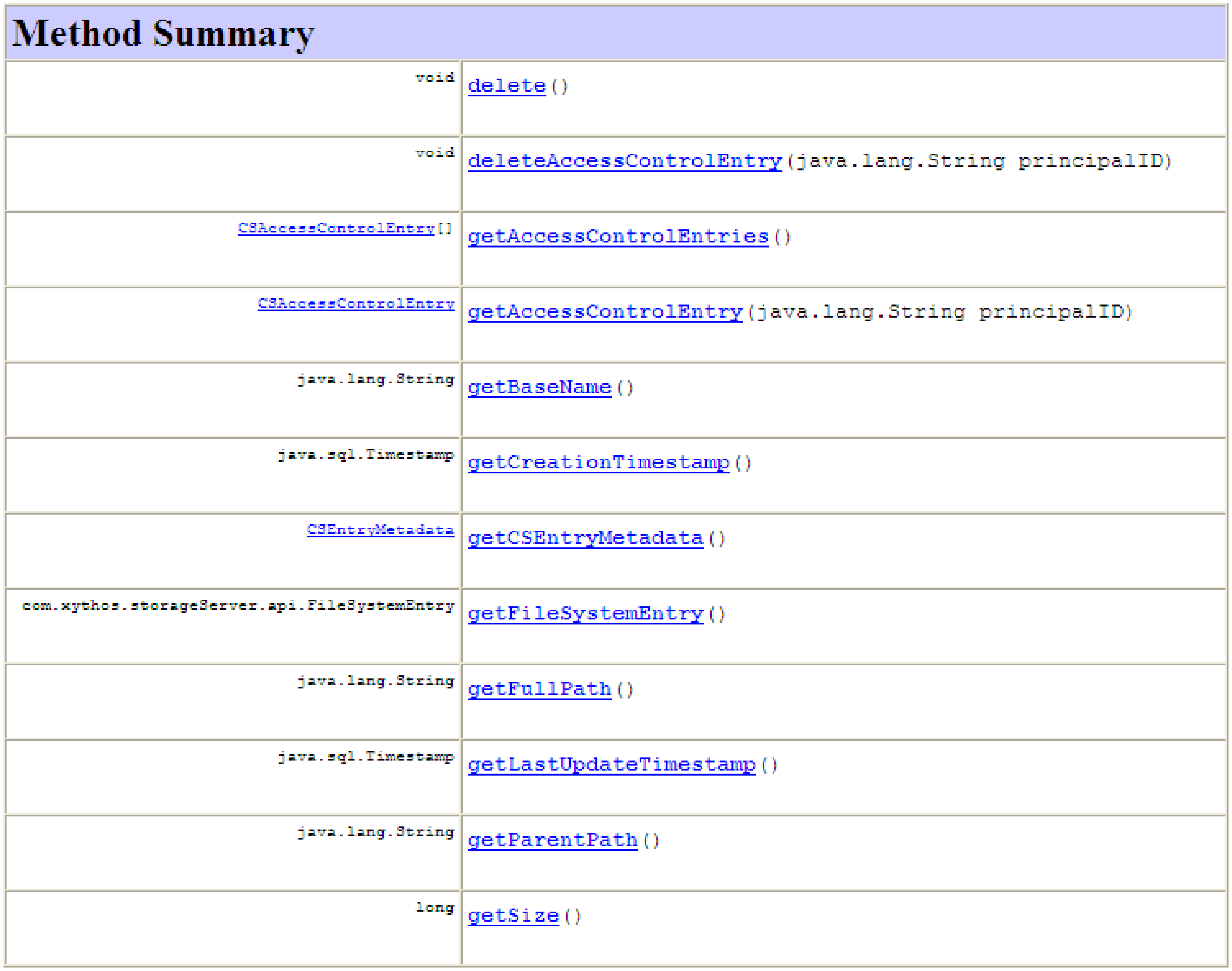
Data Object Sub-Packages (cont.)

`lrngobj`	Contains objects for interacting with the Learning Object Catalog
`metadata`	Contains classes for manipulating metadata templates
`portfolio`	Contains classes for interacting with the Portfolio system
`workflow`	Contains classes for interacting with the Workflow system

Key Data Objects

Entity	Description
`CSEntry`	This is the base class for objects (files and folders) in the filesystem. It provides methods for managing access control entries (permissions), location, size, and metadata associated with the object. See package `blackboard.cms.filesystem`
`CSFile`	This class represents files within the filesystem. It provides the method for getting and setting a file’s content, as well as the file’s content-type. See package `blackboard.cms.filesystem`

Javadoc Example for CSEntry

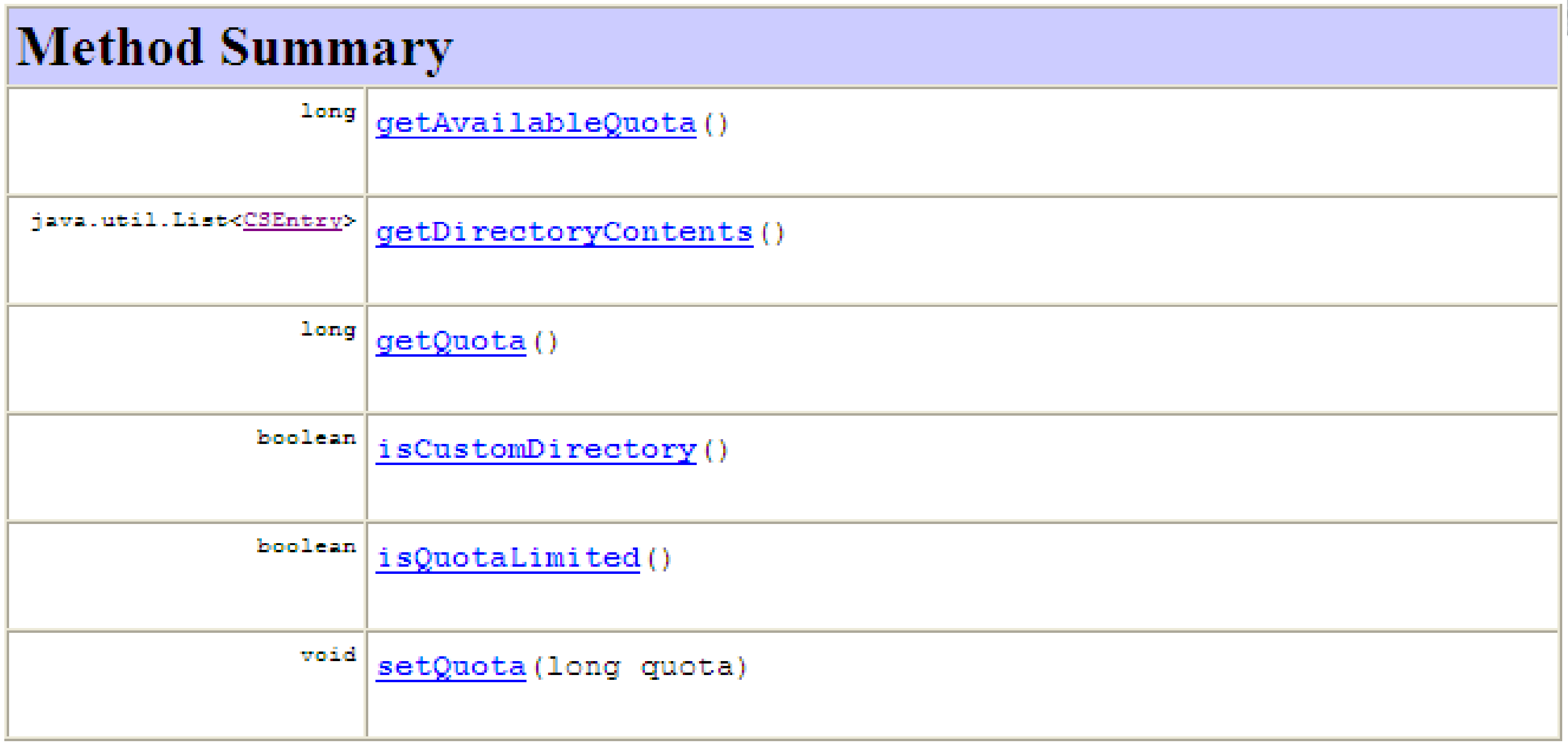


Method Summary	
void	delete()
void	deleteAccessControlEntry(java.lang.String principalID)
CSAccessControlEntry[]	getAccessControlEntries()
CSAccessControlEntry	getAccessControlEntry(java.lang.String principalID)
java.lang.String	getBaseName()
java.sql.Timestamp	getCreationTimestamp()
CSEntryMetadata	getCSEntryMetadata()
com.xythos.storageServer.api.FileSystemEntry	getFileSystemEntry()
java.lang.String	getFullPath()
java.sql.Timestamp	getLastUpdateTimestamp()
java.lang.String	getParentPath()
long	getSize()

Key Data Objects (cont.)

`CSDirectory`	This class represents a folder in the filesystem. Methods are available for managing quotas for each folder. These quotas are the total space allotted by the owner for all files and sub-folder contained within the folder. See package `blackboard.cms.filesystem`

Javadoc Example for CSDirectory



Method Summary

long	getAvailableQuota()
java.util.List<CSEntry>	getDirectoryContents()
long	getQuota()
boolean	isCustomDirectory()
boolean	isQuotaLimited()
void	setQuota(long quota)

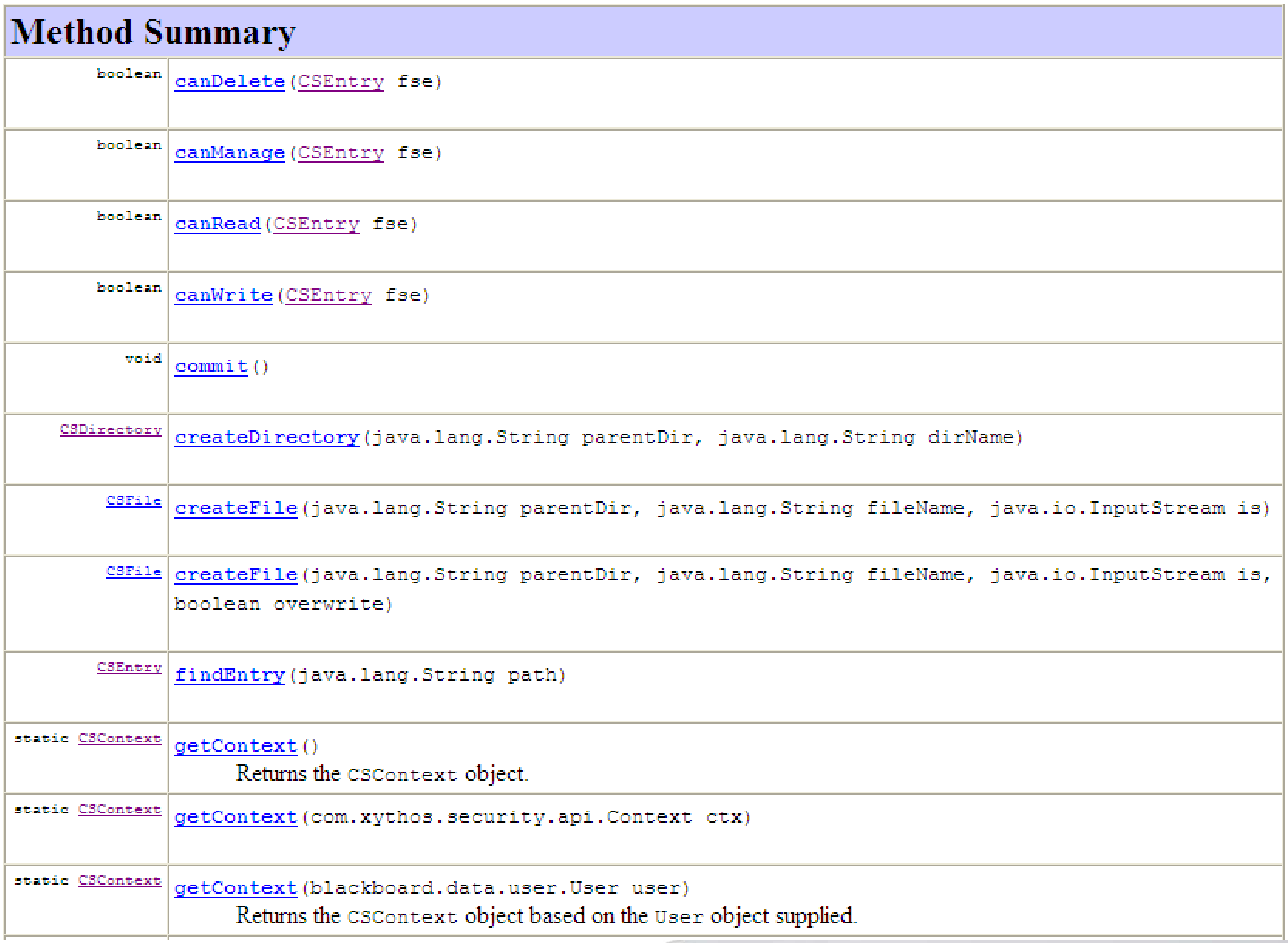
Key Data Objects (cont.)

`CSCustomDirectory`	This class encapsulates “Special” directories in the filesystem. More specifically, the directory contents of a CSCustomDirectory cannot be changed, as they are managed explicitly by Blackboard Content System. For instance, the /users, /courses directories are type CSCustomDirectory See package `blackboard.cms.filesystem`

Key Data Objects (cont.)

`CSContext`	This key object enforces filesystem security constraints (permissions) on files and folders. It also provides transaction management capabilities. A CSContext is the top level state manager for a series of Blackboard Content Management API calls. For every set of actions grouped together into one transaction, a new CSContext is created and used to either commit or rollback the work performed. See package `blackboard.cms.filesystem`

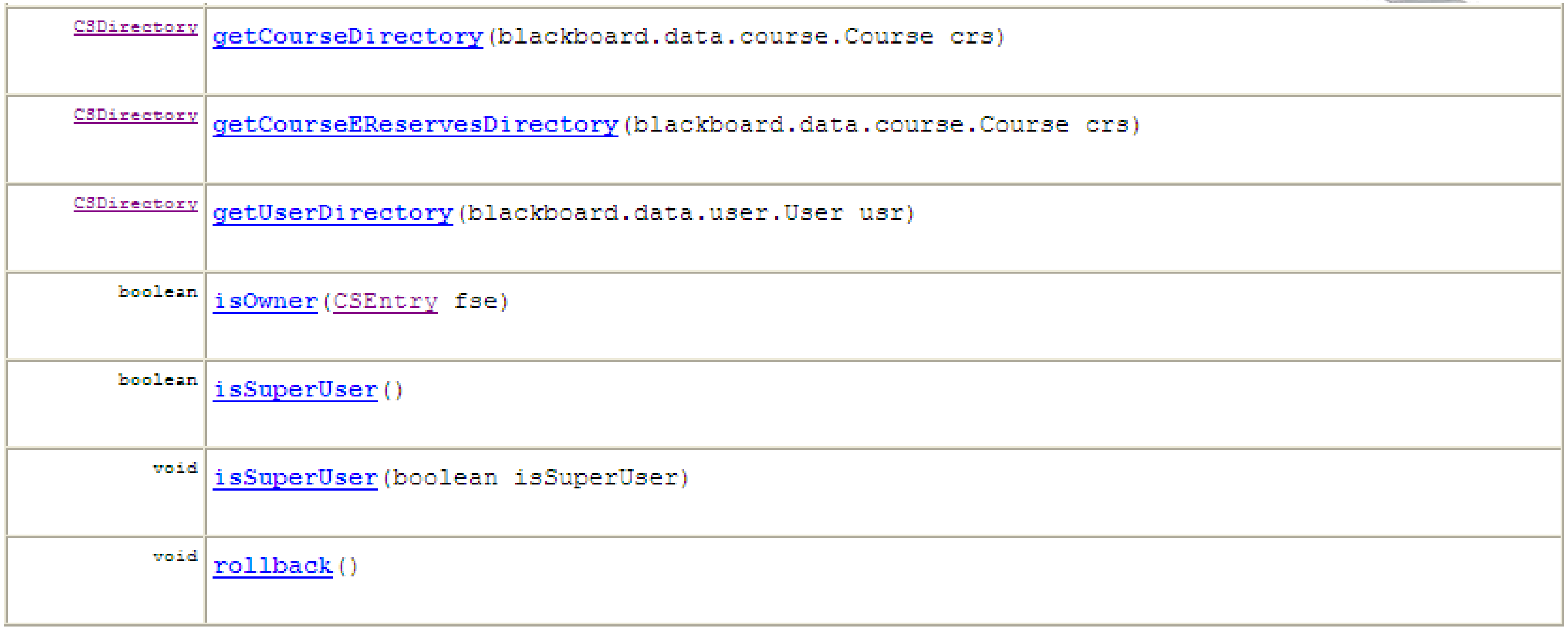
Javadoc Example for CSContext



Method Summary

boolean	canDelete(CSEntry fse)
boolean	canManage(CSEntry fse)
boolean	canRead(CSEntry fse)
boolean	canWrite(CSEntry fse)
void	commit()
CSDirectory	createDirectory(java.lang.String parentDir, java.lang.String dirName)
CSFile	createFile(java.lang.String parentDir, java.lang.String fileName, java.io.InputStream is)
CSFile	createFile(java.lang.String parentDir, java.lang.String fileName, java.io.InputStream is, boolean overwrite)
CSEntry	findEntry(java.lang.String path)
static CSContext	getContext() Returns the `CSContext` object.
static CSContext	getContext(com.xythos.security.api.Context ctx)
static CSContext	getContext(blackboard.data.user.User user) Returns the `CSContext` object based on the `User` object supplied.

Javadoc Example for CSContext (cont.)



CSDirectory	getCourseDirectory(blackboard.data.course.Course crs)
CSDirectory	getCourseEReservesDirectory(blackboard.data.course.Course crs)
CSDirectory	getUserDirectory(blackboard.data.user.User usr)
boolean	isOwner(CSEntry fse)
boolean	isSuperUser()
void	isSuperUser(boolean isSuperUser)
void	rollback()

Key Data Objects (cont.)

`CSAccessControlEntry`	The CSAccessControlEntry class provides a representation of an Access Control Entry within the Blackboard Content Management module. An object exposing the individual permissions of a file/folder for a specific user or list of users. Methods exist to both read and set the permissions. See package `blackboard.cms.filesystem`

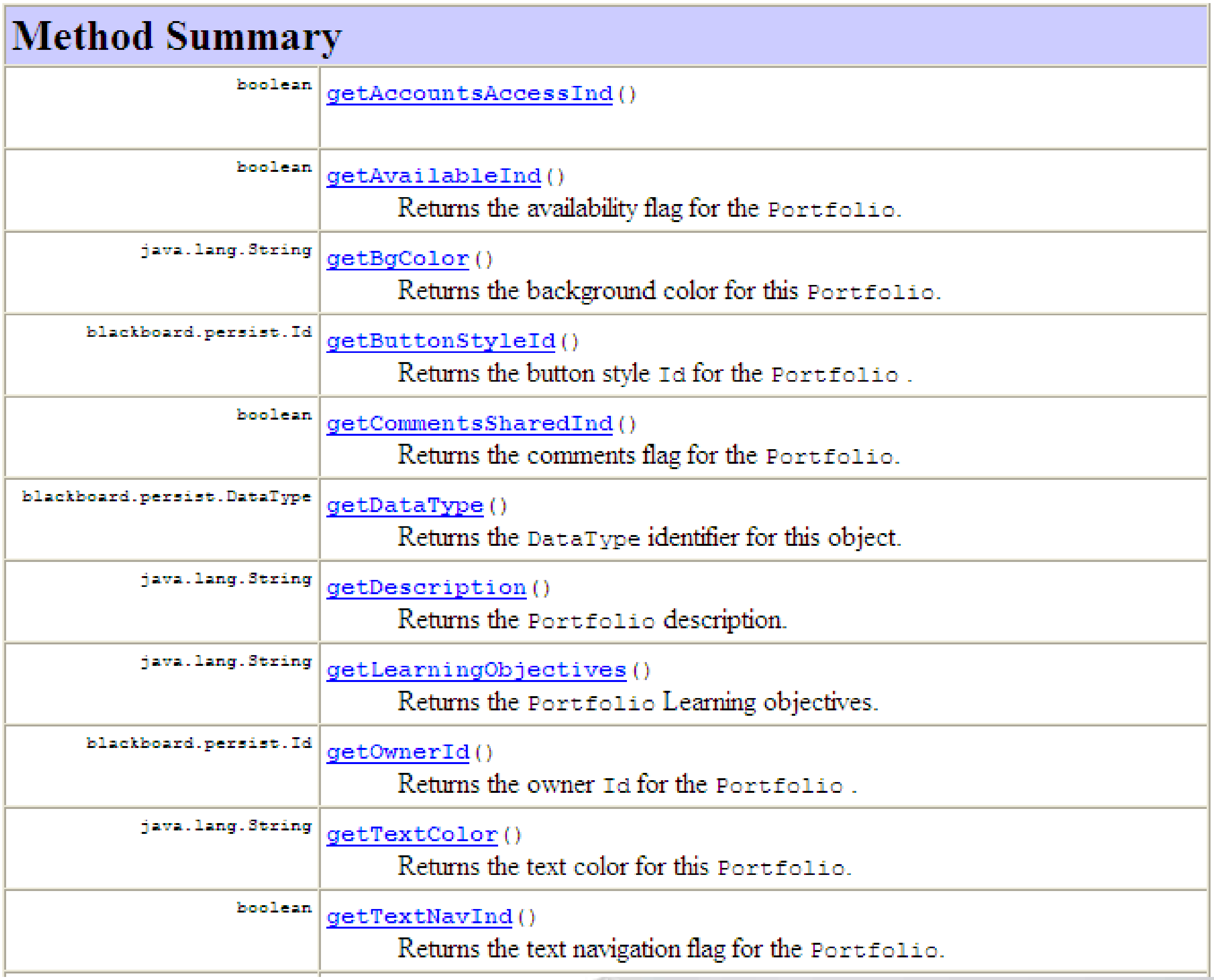
Key Data Objects (cont.)

`LOItem`	The LOItem class provides a representation of a Learning Object Item within the Blackboard Content Management module. Learning Object Items usually refer to a physical file or folder within the Blackboard Content Collection, with additional metadata and status information related as metadata. See package `blackboard.cms.lrnobj`
`LOCategory`	The LOCategory class provides a representation of a Learning Object category within the Blackboard application. See package `blackboard.cms.lrnobj`

Key Data Objects (cont.)

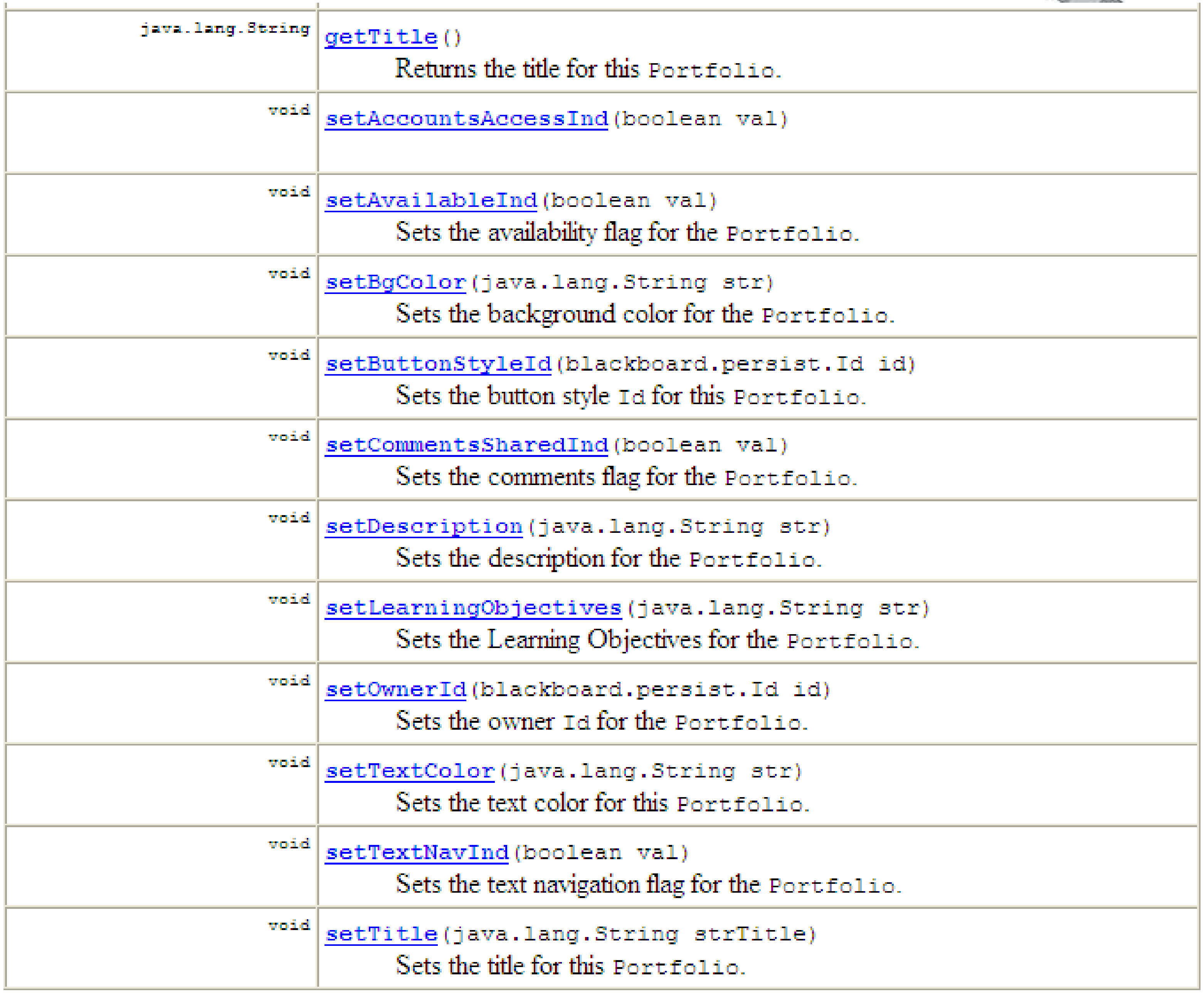
`Portfolio`	The Portfolio class provides a representation of a Portfolio within the Blackboard application. See package `blackboard.cms.portfolio`
`Workflow`	The Workflow class provides a representation of a Workflow within the Blackboard application. A Workflow must have a type associated with it. See package `blackboard.cms.workflow`

Javadoc Example for Portfolio



Method Summary	
boolean	getAccountsAccessInd()
boolean	getAvailableInd() Returns the availability flag for the Portfolio.
java.lang.String	getBgColor() Returns the background color for this Portfolio.
blackboard.persist.Id	getButtonStyleId() Returns the button style Id for the Portfolio .
boolean	getCommentsSharedInd() Returns the comments flag for the Portfolio.
blackboard.persist.DataType	getDataType() Returns the DataType identifier for this object.
java.lang.String	getDescription() Returns the Portfolio description.
java.lang.String	getLearningObjectives() Returns the Portfolio Learning objectives.
blackboard.persist.Id	getOwnerId() Returns the owner Id for the Portfolio .
java.lang.String	getTextColor() Returns the text color for this Portfolio.
boolean	getTextNavInd() Returns the text navigation flag for the Portfolio.

Javadoc Example for Portfolio (cont.)



java.lang.String	getTitle() Returns the title for this Portfolio.
void	setAccountsAccessInd(boolean val)
void	setAvailableInd(boolean val) Sets the availability flag for the Portfolio.
void	setBgColor(java.lang.String str) Sets the background color for the Portfolio.
void	setButtonStyleId(blackboard.persist.Id id) Sets the button style Id for this Portfolio.
void	setCommentsSharedInd(boolean val) Sets the comments flag for the Portfolio.
void	setDescription(java.lang.String str) Sets the description for the Portfolio.
void	setLearningObjectives(java.lang.String str) Sets the Learning Objectives for the Portfolio.
void	setOwnerId(blackboard.persist.Id id) Sets the owner Id for the Portfolio.
void	setTextColor(java.lang.String str) Sets the text color for this Portfolio.
void	setTextNavInd(boolean val) Sets the text navigation flag for the Portfolio.
void	setTitle(java.lang.String strTitle) Sets the title for this Portfolio.

Data Objects “Above” the Filesystem

- The classes in the `blackboard.cms.filesystem` packages can be thought of as the core APIs for manipulating base filesystem objects (these classes basically cover the Xythos APIs)
- The data objects found in the other packages (i.e. lrnobj, metadata, portfolio, etc) are built on top of the core filesystem APIs, and more closely follow the pattern of the other data objects found in the Blackboard Learn APIs, which means:
  - These objects subclass `blackboard.data.BbObject`
  - These objects expose the *getId()* method
  - These objects can be loaded and deleted by *Id*

Instantiating a “Compound” Data Object; example:

- The following demonstrates how to create a Portfolio object.
- **`Portfolio portTest = new Portfolio();`**
- The result of the above is a Portfolio object with default values for all of its attributes. To create a more useful Portfolio, simply call the appropriate “*set*” methods to override the default values.

Persistence Managers & “Compound” Data Objects

- The Blackboard Learn API typically makes use of Persistence Managers to manipulate (loading, deleting, saving) data objects.
- Similarly, each of the “compound”, Content Management data objects (i.e. Portfolio, Metadata Template, Learning Objects, Bookmarks, Workflows) has an associated Manager.
- For example, Portfolio has a corresponding PortfolioManager class, with the methods:
  - *deleteById()*
  - *loadById()*
  - *loadAccessibleByCourseId()*
  - *loadByOwnerId()*
  - *save(),* etc…
- These manager methods are static, so they can be accessed directly.

DEV CON '09 @BbWorld

Example of Using IDs, Loaders, and Persisters

- For example, to load a Portfolio, make a change to it, and then save it:

```
Portfolio portLoaded =
  PortfolioManager.loadById(portId);
portLoaded.setTitle(“Updated Title”);
PortfolioManager.save(portLoaded);
```

Using “Core” Content Management, File System Objects

- The “core” filesystem objects work in a slightly different way than the other Blackboard Learn data objects.
- To access “core” filesystem objects (CSEntry, CSFile, CSDirectory), you must first instantiate a Content System Context (CSContext).
- The CSContext is used to perform transaction management, as well as enforce permissions checking.

The CSContext Object

- The CSContext object is the top-level state manager for a series of Blackboard Content Management API calls.
- Instantiate a new CSContext for every set of actions grouped together into one transaction.
- A CSContext instance manages the transactional state for that request/transaction, and changes are either committed or rolled back depending on the business logic implemented by the API developer (e.g. commit on success, rollback on failure/exception)
- Each CSContext is associated with a specific user, and as such, permissions checking can be performed via CSContext calls.

The CSContext Object (cont.)

- It is extremely important when using the CSContext object to ensure that you properly use a try{},catch{},finally{} block to commit() or rollback() each transaction. ***Not doing so properly may cause connection leaks, deadlocks, performance degradation and other unexpected behavior.***
- It is recommended to always use **CSContext** in the following way:

```
CSContext ctxCS=null;
try
{
  ctxCS = CSContext.getContext();
  // your code here
}
catch (Exception e)
{
  ctxCS.rollback();
  ctxCS=null;
}
finally
{
  if (ctxCS!=null) {
    ctxCS.commit();
  }
}
```

Another Recommended Pattern for Using CSContext

```
CSContext ctxCS=null;
try
{
  try {
    ctxCS = CSContext.getContext();
    // your code here
    …
    ctxCS.commit(); //or rollback() if read-only
    ctxCS=null;
  }
  finally
  {
    if (ctxCS!=null) {
      ctxCS.rollback();
    }
  }
catch (Exception e)
{
  //do error handling or logging here
}
```

More on CSContext

- The *CSContext.getContext()* method instantiates a CSContext that is tied to the `blackboard.data.user.User` associated with the current `blackboard.platform.context.Context` (presumably, tied to the current request)
- Use *CSContext.getContext(blackboard.data.user.User user)* to instantiate a CSContext tied to “a specific user.”
- The User tied to the CSContext will be used in checking/enforcing permissions when attempting to operate on the **CSEntry** objects that are loaded via the CSContext.

Setting “Superuser” on CSContext

- For some operations, the standard User-based security enforced by CSContext may be inadequate to perform some desired operations. In that case, use the *CSContext.isSuperUser()* method to “supercharge” the User’s privileges
  - Passing in “true” to this method will enhance the privileges of the User associated with the CSContext to be those of a “superuser”.
  - Note that the operations performed using the CSContext are still considered to be performed by the User tied to the CSContext upon instantiation.
  - Take care to call *isSuperUser()* again, passing in “false” after the privileged operations are performed so that other, additional operations are not also performed using the “superuser” privileges.

Using CSContext w/ File System Objects

- An instantiated **CSContext** object can be used to create new **CSFiles** and **CSDirectories** (sub classes of the **CSEntry** object).
- It is also used to load existing files and directories using the *findEntry()* method.
- CSContext is further used to check permissions on a specific CSEntry, for the User associated with the CSContext; i.e. *canRead(), canWrite(), canManage(), canDelete()*.
- **CSContext** is also used to save (commit) changes made to CSEntry objects back to the filesystem

Example: Using CSContext to Manipulate CSEntry Objects

- Goal: Write a program that will load a specific Content Collection folder (CSDirectory) and print out the filenames of the contents (filenames) of that folder’s children.
- Preconditions
  - The User associated with the instantiated CSContext has read access to the folder and its contents.

Example: Using CSContext to Manipulate CSEntry Objects (cont.)

```
// Loads the CSDirectory object /users/jbarone
CSDirectory csDirMyUserDir =
   ctxCS.findEntry("/users/jbarone");
// Gets the contents of the directory as a List
List listDirContents =
   csDirMyUserDir.getDirectoryContents();
// Iterate through the list and print out information on
   the contents
Iterator iterDirContents = listDirContents.iterator();
while (iterDirContents.hasNext())
{
  // the List of a folder’s contents contains CSEntry
   items (which can be other folders or files)
  CSEntry csEntryItem = (CSEntry) iterDirContents.next();
  out.println(“Hello there, I have found: “
   +csEntryItem.getFullPath());
}
```

Special Loaders for Course Folders, User Folders, etc…

- CSContext also does some special loading for course, user, and eReserve folders. These “loaders” (getters) will return **CSDirectory** objects when you pass in the corresponding User or Course.
- For example, to load a Course folder in the Blackboard Content Collection for a specific *blackboard.data.course.Course* object, call the following:
  - CSDirectory courseDir =
    ctxCS.getCourseDirectory(theCourse);

Understanding Permissions & Principals

- The access control system for files and folders in the Blackboard Content Management module uses the concept of <u>Principals tied to Permissions</u>.
- A “Principal” is an entity with the base class of **CSPrincipal**, and essentially represents a user or group.
- This interface defines the concept of a Principal, encompassing any entity to which filesystem access permissions may be granted.
- Every Principal must be uniquely identified by a principalID String.
- All Blackboard Content Management user and group objects extend the Principal object.
- These objects are found in the *blackboard.cms.filesystem.security* package.

Understanding Permissions & Principals (cont.)

- Subclasses of *CSPrincipal* include:
  - **CourseGroupPrincipal, CoursePrincipal, CourseRolePrincipal, PortalRolePrincipal, PortfolioPrincipal, SystemRolePrincipal, UserPrincipal**.
  - These classes encapsulate the primary ways users are associated with permissions.
- As and example; suppose we want to give all Students in the “Math10” Course permission to “Read” some content in a Content Collection folder:
  - You must first retrieve the **CSEntry** that corresponds to the content (file or folder) to which you wish to grant access…
  - …and then determine the **CourseRolePrincipalID** used to represent the class of Users associated with a specific Role in a Course.

Understanding Permissions & Principals (cont.)

```
// Load the file
CSEntry csFileToChange =
   ctxCS.findEntry("/user/jbarone/foo.txt");
// determine the principalID corresponding to the Course
   Role
String strCourseRolePrincipal =
   CourseRolePrincipal.cacluatePrincipalID(courseMat
   h101, courseRoleofStudent);
// get the Access Control Entry for that Principal ID
   (even if it does not exist, it will be created as
   "empty")
CSAccessControlEntry csACE =
   csFileToChange.getAccessControlEntry(strCourseRol
   ePrincipal);
// then update the permission for "read"
csACE.updatePermission(true, false, false, false);
```

(HttpServlet)Request Utility Classes for Specific UI Hooks

- Content Collection Action Bar Building Blocks
  - To get a list of the files/folders selected on the previous listing page.

```
import blackboard.cms.servlet.CSActionRequest;
CSActionRequest actionReq =
   new CSActionRequest( request, response,
                        application );
List<String> selectedFiles =
   actionReq.getSelectedPaths();
```

(HttpServlet)Request Utility Classes for Specific UI Hooks (cont.)

- Modify Content Collection File/Folder Building Block

```
import blackboard.cms.servlet.CSModifyEntryRequest;

CSModifyEntryRequest modifyEntryReq =
   new CSModifyEntryRequest( request );
CSEntry csEntry =
   ctxCS.findEntry( modifyEntryReq.getPath() );
```

- Manage Portfolio Building Block

```
import blackboard.cms.servlet.CSManagePortfolioRequest;

CSManagePortfolioRequest portfolioReq =
    new CSManagePortfolioRequest( request );

Portfolio portfolio = portfolioReq.getPortfolio();
```

Content Management-specific Packaging and Manifest File Settings

- bb-manifest.xml
  - **<csversion>**
    - Element to define dependency on a specific *Blackboard Content Management* module version. The <csversion> 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 Management module present. Possible values are “fail” and “warn”. Not Required. Default value: “warn”.

Example of <csversion> in bb-manifest.xml

```
<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>
…
```

Example of UI Hooks in CS in bb-manifest.xml

```
<links>
  <link>
    <type value="cs_tool"/>
    <name value="“Your Tool Here"/>
    <url value="tool.jsp" />
    <description value="“A most excellent Building Block” />
  </link>
  <link>
    <type value="cs_action"/>
    <name value="“Your Tool Here"/>
    <url value="“actionbar.jsp" />
    <description value="“You can place Building Blocks on the action
     bar!" />
  </link>
   …
```

Example of UI Hooks in CS in bb-manifest.xml (cont.)

```
…
<link>
  <type value="cs_modify_file"/>
  <name value=“Your Tool Here"/>
  <url value="modifyfile.jsp" />
  <description value=“You can place Building Blocks on the modify
   pages!" />
</link>
<link>
  <type value="cs_modify_folder"/>
  <name value=“Your Tool Here"/>
  <url value="modifyfolder.jsp" />
  <description value=“You can place Building Blocks on the modify
   pages!" />
</link>
…
```

Example of UI Hooks in CS in bb-manifest.xml (cont.)

```
  …
  <link>
    <type value="cs_manage_portfolio"/>
    <name value=“Your Tool Here"/>
    <url value=“manageportfolio.jsp" />
    <description value=“The power of Building Blocks within Portfolios!"
     />
  </link>
  <link>
    <type value="cs_my_portfolios"/>
    <name value=“Your Tool Here"/>
    <url value=“myportfolios.jsp" />
    <description value=“The power of Building Blocks within Portfolios!"
     />
  </link>
</links>
```

Permissions Needed in bb-manifest.xml

```
<permission type="attribute" name="user.*" actions="get,set" />
<permission type=“persist" name=“Content" actions=“persist" />
<permission type=“socket" name=“*" actions=“connect" />
```

What is Xythos?

- Xythos has been a Blackboard subsidiary since end of November 2007.
- Xythos develops a platform and applications that provides simplified content management services for the enterprise.
  - Extensible platform with full-featured APIs.
  - Highly scalable and suitable for supporting enterprise applications.
  - Support for the WebDAV protocol; “basic”, Delta-V, ACLs, DASL
  - Enterprise and client applications:
    - Xythos Digital Locker → Digital Locker for Blackboard Learn
    - Xythos Enterprise Document Management Suite → Enterprise Document Management Suite for Blackboard Learn
    - Xythos Drive

What is Xythos? (cont.)

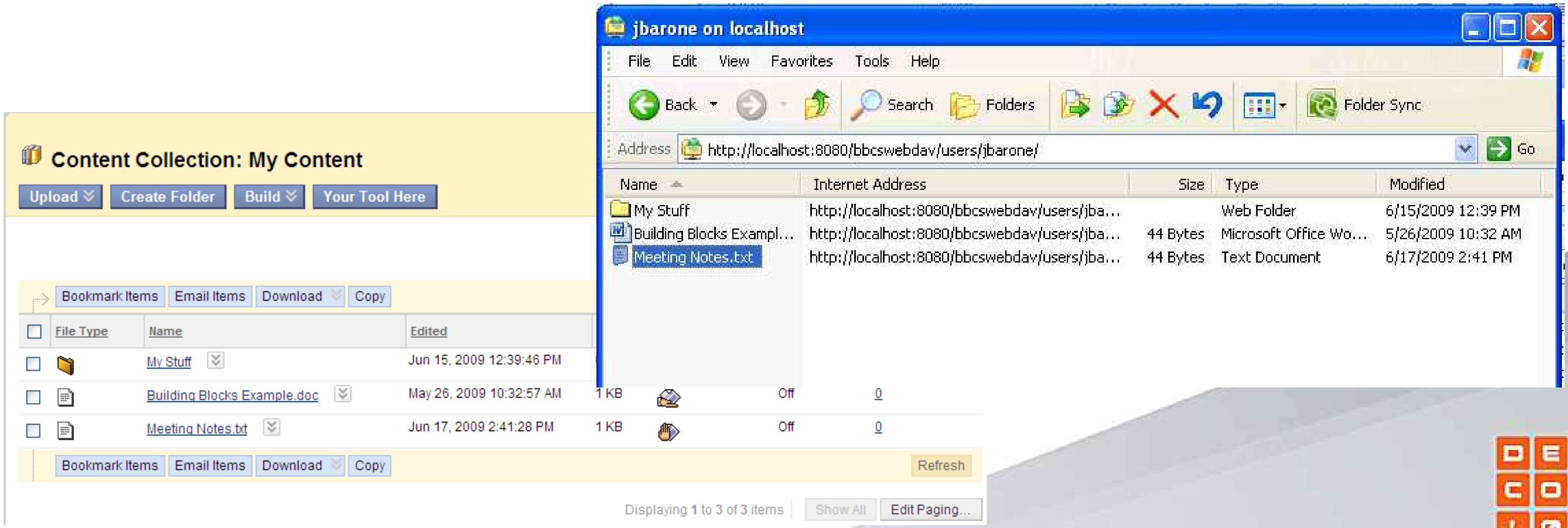
- The Content Management capabilities exposed via the Content Collection UI within Blackboard Learn use the Xythos engine as the content repository
  - Most of the classes exposed in the `blackboard.cms.filesystem` packages directly cover Xythos APIs
    - This includes the CSContext class, which covers the `com.xythos.security.api.Context` class.

What is WebDAV?

- Web(-based) Distributed Authoring and Versioning (WebDAV) is an extension to the HTTP protocol that provides for file management capabilities over the web.
  - Create/modify files and folders
  - Version files
  - View/set permissions
  - View/set meta-data
  - Search for files via file meta-data and/or content
- The Xythos platform is a WebDAV-enabled content repository, and by extension, so is the Content Management module within Blackboard Learn

What is WebDAV? (cont.)

- Why is this useful?
  - WebDAV-enabled OS' and client applications allow users to access distributed content via the native applications to which they are accustomed (e.g. Microsoft Windows, OS X, Xythos Drive)
  - A wide range of new and interesting applications can be developed on top of distributed content repositories.



Address http://localhost:8080/bbcswebdav/users/jbarone/

Name	Internet Address	Size	Type	Modified
My Stuff	http://localhost:8080/bbcswebdav/users/jba...		Web Folder	6/15/2009 12:39 PM
Building Blocks Exampl...	http://localhost:8080/bbcswebdav/users/jba...	44 Bytes	Microsoft Office Wo...	5/26/2009 10:32 AM
Meeting Notes.txt	http://localhost:8080/bbcswebdav/users/jba...	44 Bytes	Text Document	6/17/2009 2:41 PM

File Type	Name	Edited				
	My Stuff	Jun 15, 2009 12:39:46 PM				
	Building Blocks Example.doc	May 26, 2009 10:32:57 AM	1 KB		Off	0
	Meeting Notes.txt	Jun 17, 2009 2:41:28 PM	1 KB		Off	0

Displaying 1 to 3 of 3 items

Thank You!   Q&A?