VBrick Enterprise Media System

VEMS Mystro® Portal Server v6.3.x

API Reference Guide

Copyright

© 2016

VBrick Systems, Inc. All rights reserved.
2121 Cooperative Way, Suite 100
Herndon, VA 20171, USA

This publication contains confidential, proprietary, and trade secret information. No part of this document may be copied, photocopied, reproduced, translated, or reduced to any machine-readable or electronic format without prior written permission from VBrick Systems, Inc. Information in this document is subject to change without notice and VBrick assumes no responsibility or liability for any errors or inaccuracies. VBrick, VBrick Systems, the VBrick logo, VEMS Mystro, StreamPlayer, and StreamPlayer Plus are trademarks or registered trademarks of VBrick Systems, Inc. in the United States and other countries. Windows Media, SharePoint, OCS and Lync are trademarked names of Microsoft Corporation in the United States and other countries. All other products or services mentioned in this document are identified by the trademarks, service marks, or product names as designated by the companies who market those products. Inquiries should be made directly to those companies. This document may also have links to third-party web pages that are beyond the control of VBrick. The presence of such links does not imply that VBrick endorses or recommends the content of any third-party web pages. VBrick acknowledges the use of third-party open source software and licenses in some VBrick products. This freely available source code is posted at http://www.vbrick.com/opensource

About VBrick Systems

Founded in 1998, VBrick Systems is a privately held company that has enjoyed rapid growth by helping our customers successfully introduce mission critical video applications across their enterprise networks. Since our founding, VBrick has been setting the standard for quality, performance and innovation in the delivery of live and stored video over IP networks—LANs, WANs and the Internet. With thousands of video appliances installed world-wide, VBrick is the recognized leader in reliable, high-performance, easy-to-use networked video solutions.

VBrick is an active participant in the development of industry standards and continues to play an influential role in the Internet Streaming Media Alliance (ISMA), the MPEG Industry Forum, and Internet2. In 1998 VBrick invented and shipped the world's first MPEG Video Network Appliance designed to provide affordable DVD-quality video across the network. Since then, VBrick's video solutions have grown to include Video on Demand, Management, Security and Access Control, Scheduling, and Rich Media Integration. VBrick solutions are successfully supporting a broad variety of applications including distance learning and training, conferencing and remote office communications, security, process monitoring, traffic monitoring, business and news feeds to the desktop, webcasting, corporate communications, collaboration, command and control, and telemedicine. VBrick serves customers in education, government, healthcare, and financial services markets among others. VBrick products are manufactured in an ISO certified manufacturing facility.

Contents

VEMS Mystro API Reference Guide

Preface . . . vii
Getting Help . . . vii
Font Conventions . . . vii
Environmental Goals . . . viii

1. VEMS Mystro API Overview

Overview . . . 1
WCF Proxy Class . . . 1
VBBaseContract Class . . . 1

2. Logging In and Logging Out

Entering and Exiting the System . . . 3
Log In . . . 3
Log Out . . . 4

3. Users

Managing Users . . . 7
Add User . . . 7
Retrieve User . . . 9
Edit User . . . 10
Delete User . . . 11

4. Groups

Managing Groups . . . 13
Add Group . . . 13
Retrieve Group . . . 14
Edit Group . . . 16
Delete Group . . . 17

5. Content Categories

Managing Content Categories . . . 19
Add Content Category . . . 19
Retrieve Content Category . . . 20
Edit Content Category . . . 21
Delete Content Category . . . 22

6. Access Control Rights

Configuring Access Control Rights . . . 25
Set Group Access Control Level Permissions for Content Categories . . . 26
Set Group Access Control Level Permissions for Individual Units of Content . . . 30

Set User Access Control Level Permissions for Content Categories 32
Set User Access Control Level Permissions for Individual Units of Content 35
Set Content as Public 38
Get Public Status of Content 39
Set Content as Private 40
Get Private Status of Content 41
Set Default Status of Content 42
Get Default Status of Content 43

7. Video Content

Adding Video Content 45
Add a Standalone Video File 45
Add a Stored Entered URL 49
Add a Live Entered URL 53

8. Search

Searching for Content 59
Clear the Content Cache 59
Search for Content 60
Search for Content Instances 63

9. Content Instances

Managing Content Instances 65
Add Content Instance 65
Edit Content Instance 67
Delete Content Instance 68

10. Play and Stop Content

Playing and Stopping Content 71
Play Content 71
Play Content Instance 73
Stop Playing Content. 75
Play Content to Set Top Box (STB) 76
Rebroadcast Content to STB. 76
Live Broadcast Content to STB 84
Tune STB to Existing Stream 93

11. Record Content

Recording Content 101
Determine if Content is Recordable 101
Start Recording Content 102
Edit Recording Metadata. 103
Stop Recording Content 106
Get Record Request Status 107
Record to VBStar 108

12. Embed and Share Video

Embedding Code & Sharing a URL 117
Embed Code 117
Share URL Code 120

VEMS Mystro API Reference Guide

Preface

This document explains how to accomplish the most commonly used tasks through the **VEMS Mystro API**. It is assumed that the reader is an experienced software developer with knowledge of Web development, the .NET 4.0 Framework, and Windows Communication Foundation (WCF). All code examples are written in C# unless otherwise noted.

Getting Help

If you need help, or more information about any topic, use the online help system. The online help is cross-referenced and searchable and can usually find the information in a few seconds. Use the tree controls in the left pane to open documents and the up and down arrows to page through them. Use the **Search** box to find specific information. Simply enter one or more words in the box and press Enter. The search results will return pages that have all of the words you entered—highlighted in yellow (Internet Explorer only). The **Search** box is not case-sensitive and does not recognize articles (a, an, the), operators (+ and –), or quotation marks. You can narrow the search by *adding* words.

If you can't find the information you need from the online help, or from your certified VBrick reseller, you can contact VBrick Support Services on the web. Support Services can usually answer your technical questions in 24 business hours or less. Also note that our publications team is committed to accurate and reliable documentation and we appreciate your feedback. If you find errors or omissions in any of our documents, please send e-mail to documentation@vbrick.com and let us know. For more information about any VBrick products, all of our product documentation is available on the web. Go to www.vbrick.com/documentation to search or download VBrick product documentation.

Font Conventions

Arial bold is used to describe dialog boxes and menu choices, for example: **Start > All Programs > VBrick**

`Courier fixed-width font` is used for scripts, code examples, or keyboard commands.

`Courier bold fixed-width font` is used for user input in scripts, code examples, or keyboard commands.

This bold black font is used to strongly emphasize important words or phrases.

Folder names and user examples in text are displayed in this sans serif font.

User input in text is displayed in this bold sans serif font.

Italics are used in text to emphasize specific words or phrases.

Environmental Goals

At VBrick, we believe that running our company with a "green" conscience is good for the environment and good for business and that environmental awareness is an important part of the value we deliver to our customers. We recognize our responsibilities to our customers, partners, and employees, and also to the communities in which we live and work. We believe that the same ethics and principles that guide our daily business decisions should be applied to the environment as well. We design superior quality, high performance, and energy-efficient products and are continually looking for ways to conserve energy and reduce waste. As a company, we look for ways to be environmentally friendly in designing our products and operating our facilities, and by choosing partners and suppliers who are committed to sustainable development. You can help by recycling batteries and other consumables and by finding new and better ways to protect and preserve our environment. If you have ideas or suggestions that will help to reinforce our commitment to these goals, please let us know.

Chapter 1

VEMS Mystro API Overview

Topics in this document

Overview 1
WCF Proxy Class 1
VBBaseContract Class 1

Overview

The VEMS Mystro API is the "brains" of an entire VEMS Mystro eco-system. The API is based on Microsoft Windows Communication Foundation (WCF) technology and can be used with a variety of front ends.

This reference guide will introduce the proxy object which acts as the intermediary between the GUI and back-end. Also, a special class called **VBBaseContract** will be briefly examined. Output from just about all API methods return **VBxxx** objects, which are all derived from the **VBBaseContract** class.

This guide will begin with the API calls to log in/log out and then progress to adding, editing and deleting users, groups, and categories. Once the dynamics of adding, editing and deleting users, groups and categories have been mastered, more complicated tasks such as granting access control rights at the user, group, category and content level can be explored. Once access control functionality is mastered, the ability to add/edit/delete video content and video content instances will be introduced. Then the ability to start/stop broadcasts, play content on set top boxes, retrieve embedded code, and finally perform video content searches will be covered.

WCF Proxy Class

The C# code examples listed throughout this guide make use of a proxy object referred to as "sll". Note that a proxy object is *not* necessary to interact with the API. The specific method used to access the API will depend on the developer's development environment. For instance, a developer can make web service calls directly from JavaScript/jQuery, or use sockets in conjunction with the SOAP::Lite module in Perl to build and parse SOAP messages.

VBBaseContract Class

All **VBxxx** objects either passed into an API method or returned from an API method are derived from the base class **VBBaseContract**.

The purpose of deriving all VBxxx classes from this base class is so that exceptions that occur on the server side can be safely communicated back to the calling method. When an exception is communicated back to the calling method an action can be taken, such as displaying an error message. All VBxxx objects have an `Exception` field. After calling an API method, this field must be examined. If the field is null then the API call completed *without*

error. If the Exception member is populated, then the API call failed. Information about the failure can be found in the `Exception` field as well as the **ExceptionLog** table in the database.

Chapter 2

Logging In and Logging Out

Topics in this document

Entering and Exiting the System 3
Log In 3
Log Out 4

Entering and Exiting the System

The VEMS Mystro API provides the ability to enter and exit the system by providing methods to Log in and Log out.

The **UserLogin** method is the very first method that *must* be called before any other interaction with the API can occur. Further, *all* API calls require a **SessionID**, which is returned from a successful call to the UserLogin method.

Log In

A user must log in to VEMS Mystro before any other interaction with the API can be performed.

Syntax:

```
UserLogin(Username, Password, ApplicationID, Host, UserLanguage)
```

Inputs:

Name	Description	Type
Username	The username of the user to log in with.	string
Password	The password of the user to log in with.	string
ApplicationID	The application ID listed in the ClientApplication table that corresponds to the desired API consumer. This information can be found by using the following query (replace the placeholder with the desired consumer): Select Application ID from ClientApplication where Name = '<Consumer>'	string
Host	The IP address or host name of the client.	string
UserLanguage	The RFC 1766 language code (Example: EN-US for United States English).	string

Output:

```
VBSession object
```

The **VBSession** object returned represents a VEMS Mystro session. The **SessionID** field of the returned VBSession object is used when calling all other API methods. Client side code must examine the `Exception` field of the VBSession object to determine if the **UserLogin** operation was successful. If the Exception field is null, the operation was successful and a valid Session ID should be present in the SessionID field. If the Exception field is not null, then the log in attempt failed.

Example Client Code:

```
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);

if (mySession.Exception != null)
{
//failure code path; do something to indicate to the user that an occur occurred
}
else
{
//success code path; continue on with client application code
}
```

Log Out

When finished using VEMS Mystro, the user should log out of the system.

Syntax:

```
UserLogout(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID that belongs to the user logging out.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the log out operation.

Example Client Code:

```
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);

…//do some work here

VBVoidData userLogoutResponse = sll.UserLogout(mySession.SessionID);

if (userLogoutResponse.Exception != null)
{
//failure code path; do something to indicate to the user that an occur occurred
}
else
{
```

```
//success code path; continue on with client application code
}
```

Chapter 3

Users

Topics in this document

Managing Users 7
Add User 7
Retrieve User 9
Edit User 10
Delete User 11

Managing Users

The VEMS Mystro API provides the methods to create, retrieve, edit and delete users. The methods used to accomplish these tasks are discussed in the following sections.

Add User

The VEMS Mystro API provides the ability to add a user to the system.

Syntax:

```
UserAdd(newUserObj, sessionID)
```

Inputs:

Name	Description	Type
newUserObj	The VBUser object representing the new user to add to the system.	VBUser
sessionID	The session ID that belongs to the user creating this new user.	string

The following properties on the newUserObj can be set prior to passing this object to the UserAdd method.

Name	Description	Type	IsRequired?
UserName	The username of the new user.	string	Y
Password	The password of the new user.	string	Y
FirstName	The first name of the new user.	string	Y
LastName	The last name of the new user.	string	Y
PIN	The Personal Identification Number of the new user.	string	N

Name	Description	Type	IsRequired?
EnumUserTypeID	The type of the new user.	int	Y
EnumUserStatusID	The status of the new user.	int	Y
IsLDAPUser	Indicates whether or not the user was imported from an LDAP group.	boolean	N

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the UserAdd operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);

//make sure login was successful, declare variables, populate variables with values
from GUI, etc.

//create user
VBUser myNewUser = new VBUser();
myNewUser.UserName = newUserUserName;
myNewUser.Password = newUserPassword;
myNewUser.PIN = newUserPIN;
myNewUser.IsLDAPUser = false;
myNewUser.EnumUserTypeID = (int)USERTYPE.InteractiveUser;
myNewUser.EnumUserStatusID = (int)USERSTATUS.Active; //Could also be set to
TempLock or PermLock

//add user
VBVoidData addUserResponse = sll.UserAdd(myNewUser, mySession.SessionID);

if (addUserResponse.Exception != null)
{
//failure code path
}
else
{
//success code path
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);

if (logOutResponse.Exception != null)
{
//failure path, could not logout
}
else
{//successfully logged out
}
```

If an exception is not returned, the new user is created. Note that this new user does not have any groups, roles, or category and content permissions assigned yet. Before this user can be modified, there must be a way to retrieve a VBUser object representing the current state of this user. This will be discussed in the next section.

Retrieve User

The VEMS Mystro API provides a couple different methods that can be used to retrieve a VBUser object representing a specific user; You may retrieve a user by ID or by name. Both methods are discussed below.

Method 1:

Syntax:

```
UserGetByUserID(targetUserID, sessionID)
```

Inputs:

Name	Description	Type
targetUserID	The user ID of the user to retrieve.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBUser object
```

The returned **VBUser** object represents the user who belongs to the **targetUserID**.

Method 2:

Syntax:

```
UserGetByUserName(targetUserName, sessionID)
```

Inputs:

Name	Description	Type
targetUserName	The user name of the user to retrieve.	string
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBUser object
```

The returned **VBUser** object represents the user who belongs to the **targetUserName**.

Example Client Code:

Method 1:

```
//login
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);

//make sure login was successful, etc.

VBUser myTargetUser = sll.UserGetByUserID(targetUserID, mySession.SessionID);

if(myTargetUser.Exception != null)
{
//failure path, user with targetUserID was not found
}
else
```

```
{
//success path
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);

if (logOutResponse.Exception != null)
{
//failure path, could not logout
}
else
{//successfully logged out
}
```

Method 2:

```
//login
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);
//make sure login was successful, etc.

VBUser myTargetUser = sll.UserGetByUserName(targetUsername, SessionID);
if(myTargetUser.Exception != null)
{
//failure path, user with targetUsername was not found
}
else
{
//success path
}
//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);

if (logOutResponse.Exception != null)
{
//failure path, could not logout
}
else
{//successfully logged out
}
```

Edit User

The VEMS Mystro API provides a method to edit a user. Once a **VBUser** object representing the target user has been retrieved, fields in the object can then be updated to reflect changes to the user. Once the fields have been updated, the *state* of the target VBUser object can be persisted to the database by calling the corresponding update method.

Syntax:

```
UserUpdate(targetUserObj, sessionID)
```

Inputs:

Name	Description	Type
targetUserObj	VBUser object representing the user to be updated.	VBUser
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the UserUpdate operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);

//make sure login was successful, etc.

//retrieve user
VBUser myTargetUser = sll.UserGetByUserName(targetUsername, mySession.SessionID);
//make sure the user was retrieved without error

//update fields that need updating
myTargetUser.FirstName = "newFirstName";
myTargetUser.LastName = "newLastName";
myTargetUser.PIN = "newPIN";

//make the update
VBVoidData updateUserResponse = sll.UserUpdate(myTargetUser, mySession.SessionID);
if (updateUserResponse.Exception != null)
{
//failure path
}
else
{
//success path
//refresh reference to myTargetUser before using it again anywhere else
myTargetUser = sll.UserGetByUserName(myTargetUser.UserName, mySession.SessionID);
}
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //deal with failure here
```

Delete User

The VEMS Mystro API provides the ability to delete users from the system.

Syntax:

```
UserDelete(targetUserObj, sessionID)
```

Inputs:

Name	Description	Type
targetUserObj	The object representing the user to be deleted.	VBUser
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the UserDelete operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);

//make sure login was successful, etc.

//retrieve user
VBUser userToDelete = sll.UserGetByUserName(targetUsername, mySession.SessionID);

//make sure user retrieval was successful

//let's delete this user
VBVoidData deleteUserResponse = sll.UserDelete(userToDelete, mySession.SessionID);

if (deleteUserResponse.Exception != null)
{
//failure path, the user was not deleted
}
else
{
//success path
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);

if (logOutResponse.Exception != null)
{
//failure path, could not logout
}
else
{//successfully logged out
}
```

Chapter 4

Groups

Topics in this document

Managing Groups. 13
Add Group. 13
Retrieve Group. 14
Edit Group. 16
Delete Group 17

Managing Groups

The VEMS Mystro API provides the methods to create, retrieve, edit and delete groups. The methods used to accomplish these tasks are discussed in the following sections.

Add Group

The VEMS Mystro API provides the ability to add a group to the system.

Syntax:

```
GroupAdd(newGroupObj, sessionID)
```

Inputs:

Name	Description	Type
newGroupObj	The object representing the group to add.	VBGroup
sessionID	The session ID of the user invoking the method.	string

The following properties on the **newGroupObj** can be set prior to passing this object to the GroupAdd method.

Name	Description	Type	IsRequired
Name	The name of the group.	string	Y
Description	A description of the group.	string	N
Users	The users that belong to the group.	List<VBUser>	N

Output:

```
VBGroup object
```

The **VBGroup** object returned represents the new Group that was just created. The client code should examine this object's `Exception` field to determine if an error occurred while attempting to create this new group.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);
//make sure login was successful, etc.

//build up a VBGroup object
VBGroup newGroupToAdd = new VBGroup();
newGroupToAdd.Name = "groupName";
newGroupToAdd.Description = "group Description";

VBGroup newGroupRet = sll.GroupAdd(newGroupToAdd, mySession.SessionID);

if (newGroupRet.Exception != null)
{
//failure path
}
else
{
//success path
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);

if (logOutResponse.Exception != null) //failure path, could not logout
```

Retrieve Group

The VEMS Mystro API provides 1 direct method to retrieve a **VBGroup** object that represents a specific group.

However, there is also another method that can be used to help developers write their own **GroupGetXX** methods. Both are discussed below.

Method 1:

Syntax:

```
GroupGet(targetGroupID, sessionID)
```

Inputs:

Name	Description	Type
targetGroupID	The group ID of the group to retrieve.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBGroup object
```

The **VBGroup** object returned represents the Group associated with the **targetGroupID**. The client side code should examine the `Exception` field to determine if an error occurred during the GroupGet operation.

Method 2:

Syntax:

```
GroupsGetAll(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBGroup>
```

The returned **VBList<VBGroup>** represents a List of *all* the Groups in the system. The client code should examine the `Exception` field to determine if an error occurred during the GroupsGetAll operation.

A developer can use this method inside of a custom **GroupGetByXX** method to search for a group based on its name, description, or some other custom criteria.

Example Client Code:

Method 1:

```
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);
//make sure login was successful, etc.

VBGroup myGroup = sll.GroupGet(targetGroupID, mySession.SessionID);
if (myGroup.Exception != null)
{
//failure path
}
else
{
//success path
}

VBVoidData logoutResponse = sll.UserLogout(mySession.SessionID);
If (logoutResponse.Exception != null) //failure code
```

Method 2:

```
//login
VBSession mySession = sll.UserLogin(Username, password, ApplicationID, ClientIP,
UserLanguage);
//make sure login was successful, etc.
string groupName = "groupNameToSearchFor";

VBGroup targetGroup = customGetGroupByName(sll, mySession.SessionID, groupName);
```

```
if (targetGroup != null)
{
//found the targetGroup
}
else
{
//could not find the targetGroup
}

VBVoidData logoutResponse = sll.UserLogout(mySession.SessionID);
if (logoutResponse.Exception != null) //Logout failure code here

private VBGroup customGetGroupByName(MaduroSLL.IMaduroSLL sll, string sessionID,
string grpName)
{
//retrieve all groups
VBList<VBGroup> allGroups = sll.GroupsGetAll(sessionID);

VBGroup targetGroup = null;
foreach(VBGroup currentGroup in allGroups.Entities)
{
if(currentGroup.Name == grpName)
{
targetGroup = currentGroup;
break;
}
}
return targetGroup;
}
```

Edit Group

The VEMS Mystro API provides the ability to edit a group in the system. Once a **VBGroup** object representing the target group has been retrieved, fields in the object can be modified to reflect changes to the group. Once the fields have been updated, the state of the target VBGroup object can be persisted to the database by calling the **GroupUpdate** method.

Syntax:

```
GroupUpdate(groupToUpdateObj, sessionID)
```

Inputs:

Name	Description	Type
groupToUpdateObj	The object representing the group to update.	VBGroup
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBGroup object
```

The **VBGroup** object returned represents the updated group. The client code should examine this object's `Exception` field to determine if an error occurred during the GroupUpdate operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful

string targetGroupName; //retrieve targetGroupName from the GUI
//get reference to this target group (the implementation of customGetGroupByName
not shown here)
VBGroup targetGroupForUpdate = customGetGroupByName(sll, mySession.SessionID,
targetGroupName);

if (targetGroupForUpdate != null)
{//modify the desired fields
targetGroupForUpdate.Name = "newName";
targetGroupForUpdate.Description = "newDescription";
}
else
{
throw new Exception("Could not find target group");
}

//make the update
targetGroupForUpdate = sll.GroupUpdate(targetGroupForUpdate, mySession.SessionID);
if (targetGroupForUpdate.Exception != null)
{
//failure path
}
else
{
//success path
}

VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //failure code path
```

Delete Group

The VEMS Mystro API provides the ability to delete a group from the system.

Syntax:

```
GroupDelete(groupToDeleteObj, sessionID)
```

Inputs:

Name	Description	Type
groupToDeleteObj	The object representing the group to be deleted.	VBGroup
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the GroupDelete operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
```

```
UserLanguage);
//verify that login was successful

string groupNameToDel; //retrieve groupNameToDelete from GUI

//get reference to this target group (the implementation of customGetGroupByName
not shown here)
VBGroup targetGroupForUpdate = customGetGroupByName(sll, mySession.SessionID,
groupNameToDel);

if (targetGroupForUpdate == null)
{
throw new Exception ("Could not find Group with Name: " + groupNameToDel);
}

//delete this group
VBVoidData deleteGroupResponse = sll.GroupDelete(targetGroupForUpdate,
mySession.SessionID);

if (deleteGroupResponse.Exception != null)
{
//failure path (the group was not deleted)
}
else
{
//success path
}

VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //failure code path here
```

Chapter 5

Content Categories

Topics in this document

Managing Content Categories 19
Add Content Category 19
Retrieve Content Category 20
Edit Content Category 21
Delete Content Category 22

Managing Content Categories

The VEMS Mystro API provides the methods to create, retrieve, edit and delete content categories. The methods used to accomplish these tasks are discussed in the following sections.

Add Content Category

The VEMS Mystro API provides the ability to add a content category to the system.

Syntax:

```
CategoryAdd(categoryToAddObj, sessionID)
```

Inputs:

Name	Description	Type
categoryToAddObj	The object representing the category to add.	VBCategory
sessionID	The session ID of the user invoking the method.	string

The following properties on the **categoryToAddObj** can be set prior to passing this object to the CategoryAdd method.

Name	Description	Type	IsRequired
Name	The name of the category.	string	Y
ParentCategoryID	The category ID of the parent category.	int	N

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the CategoryAdd operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful

string categoryNameToAdd; //get this value from the GUI

VBCategory categoryToAdd = new VBCategory();
categoryToAdd.Name = categoryNameToAdd;

VBVoidData addCategoryResponse = sll.CategoryAdd(categoryToAdd,
mySession.SessionID);

if (addCategoryResponse.Exception != null)
{
//failure path
}
else
{
//success path
}

VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logoutResponse.Exception != null) //failure code path here
```

Retrieve Content Category

The VEMS Mystro API does *not* provide a method that directly retrieves a specific category by name or ID. However, the API does provide a method that retrieves all categories in the system. A developer can use this method to create a custom **GetCategoryByXX** method.

Syntax:

```
CategoriesGetAll(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBCategory>
```

The **VBList<VBCategory>** object returned from the method is a List of VBCategory objects representing all of the Categories in the system. The `Exception` field of this object should be examined to determine if an error occurred during the CategoriesGetAll operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
```

```
UserLanguage);
//verify that login was successful

string categoryNameToSearch; //retrieve value from GUI

VBCategory targetCatgory = GetCategoryByName(sll, mySession.SessionID,
categoryNameToSearch);

if (targetCategory == null) throw new Exception("Cannot find category:" +
categoryNameToSearch);

//continue working with category

VBVoidData logoutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //failure code path here

//returns the first category with a name matching catName
private VBCategory GetCategoryByName(MaduroSLL.IMaduroSLL sll, string sessionID,
string catName)
{
VBList<VBCategory> allCategories = sll.CategoriesGetAll(sessionID);

VBCategory targetCategory;
foreach(VBCategory currentCat in allCategories.Entities)
{
if (currentCat.Name == catName)
{
targetCategory = currentCat;
break;
}
}

return targetCategory;
}
```

Edit Content Category

The VEMS Mystro API provides the ability to edit a Content Category in the system. Once a **VBCategory** object representing the target category has been retrieved, fields in the object can be modified to reflect changes to the category. Once the fields have been updated, the state of the VBCategory object can then be persisted to the database by calling the **CategoryUpdate** method.

Syntax:

```
CategoryUpdate(categoryToUpdateObj, sessionID)
```

Inputs:

Name	Description	Type
categoryToUpdateObj	The object representing the category to update.	VBCategory
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the CategoryUpdate operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful

string categoryNameToSearch; //retrieve value from GUI
//Note: GetCategoryByName() implementation not shown here for clarity
VBCategory targetCategory = GetCategoryByName(sll, mySession.SessionID,
categoryNameToSearch);

//modify the name of the category
targetCategory.Name = "some New Name";

VBVoidData updateCategoryResponse = sll.CategoryUpdate(targetCategory,
mySession.SessionID);

if (updateCategoryResponse.Exception != null)
{
//failure path
}
else
{
//success path
}

VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
//failure path here
}
else
{
//success path here
}
```

Delete Content Category

The VEMS Mystro API provides the ability to delete a content category from the system.

Syntax:

```
CategoryDelete(categoryToDeleteObj, sessionID, isCascadeDelete)
```

Inputs:

Name	Description	Type
categoryToDeleteObj	The object representing the category to be deleted.	VBCategory
sessionID	The session ID of the user invoking the method.	string

Name	Description	Type
isCascadeDelete	Flag indicating whether or not the delete should cascade to all child categories of the category being deleted. (This is usually set to true)	boolean

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the CategoryDelete operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful

string categoryNameToSearch; //retrieve value from GUI
//Note: GetCategoryByName() implementation not shown here for clarity
VBCategory targetCategory = GetCategoryByName(sll, mySession.SessionID,
categoryNameToSearch);

VBVoidData deleteCategoryResponse = sll.CategoryDelete(targetCategory,
mySession.SessionID, true);

if (deleteCategoryResponse.Exception != null)
{
//failure path
}
else
{
//success path
}

VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
//failure path
}
else
{
//success path
}
```

Chapter 6

Access Control Rights

Topics in this document

Configuring Access Control Rights 25
Set Group Access Control Level Permissions for Content Categories 26
Set Group Access Control Level Permissions for Individual Units of Content 30
Set User Access Control Level Permissions for Content Categories. 32
Set User Access Control Level Permissions for Individual Units of Content 35
Set Content as Public 38
Get Public Status of Content. 39
Set Content as Private 40
Get Private Status of Content 41
Set Default Status of Content 42
Get Default Status of Content. 43

Configuring Access Control Rights

Access control permissions for users and groups may be configured at the content category and individual content levels. Before configuring access control permissions, you must first understand the different access control levels and how they work.

There are 6 access control levels available. These levels are:

- Add
- View
- Share
- Edit
- Delete
- Admin

All access control permissions set at the content category level propagate down to each individual unit of content contained therein, unless there are explicit content permissions set for individual units of content. Individual content permissions supersede permissions set at the category level.

The **Add** access control level means that the grantee (user or group) has the ability to add content to a particular content category.

The **View** access control level means that the grantee has the ability to view a content category and view all content contained therein (unless this permission is overridden at the individual content level).

The **Share** access control level means that the grantee has the ability to share the content category and all content contained therein. This access control level is *always* set to the same value as the **View** access control level.

The **Edit** access control level means that the grantee has the ability to edit a content category and edit any of the content contained therein (unless this permission is overridden at the individual content level).

The **Delete** access control level means that the grantee has the ability to delete a content category and delete any content contained therein (unless this permission is overridden at the individual content level). When a unit of content is deleted from a category, it is not physically deleted from the system; it is just no longer a member of that category and becomes a member of the "Uncategorized" pseudo category.

The **Admin** access control level means that the grantee has the ability to modify the access control rights of a content category and that of all content contained therein (unless this permission is overridden at the individual content level).

The access control rights described above can be applied to content categories and individual units of content for users and groups. The highest priority are the permissions set at *individual* units of content; these permissions will supersede any permissions set at the category level. These individual content permissions for a user and his/her group(s) are unioned together and applied. If no individual content permissions are set, then the content category permissions that belong to the user and his group(s) are unioned and applied.

In addition to the 6 access control levels examined here, an individual unit of content can have content permissions set to **Default**, **Public** or **Private**.

Content marked as **Default** means that the access control permissions discussed above are in effect.

Content marked as **Public** means that every user in the system is implicitly granted **View** access only, even if a user is explicitly denied View access.

Content marked as **Private** means that the content does not inherit any permissions set at the content category level. Only the individual content permissions set for a user and his group(s) (if any exist) are unioned and applied.

Set Group Access Control Level Permissions for Content Categories

The VEMS Mystro API provides the ability to retrieve and update the access control level permissions for content categories and individual units of content contained therein at the group level. Setting access control level permissions for content categories will be explored first, followed by setting access control permissions for individual units of content in 20.2. There are 3 methods involved with the retrieval and update of content category access control permissions. Each method will be explained below and then a code sample demonstrating their use will be presented and explained.

Method 1:

Syntax:

```
CategoriesGetAllWithGroupPermissions(targetGroupID, sessionID)
```

Inputs:

Name	Description	Type
targetGroupID	The group ID of the group whose category permissions are being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBCategory>
```

The **VBList<VBCategory>** object returned is a **VBList** of **VBCategory** objects. Within each VBCategory object there is a `CategoryPermission` field which will contain a **VBCategoryPermission** object, if and only if there are permissions assigned, otherwise this field is null. Also, examine the `Exception` field to determine if an error occurred during the CategoriesGetAllWithGroupPermissions operation.

Method 2:

Syntax:

```
ContentCategoryPermissionsDefaultGroupGet(targetGroupID, sessionID)
```

Inputs:

Name	Description	Type
targetGroupID	The group ID of the group whose default category permissions are being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBDefaultContentCategoryPermissions object
```

The **VBDefaultContentCategoryPermissions** object returned contains the default content category permissions assigned to the group. These defaults are determined by unioning all the category permissions of the feature sets that the group belongs too. Examine the `Exception` field to determine if an error occurred during the ContentCategoryPermissionsDefaultGroupGet operation.

Method 3:

Syntax:

```
CategoryContentPermissionsUpdateByGroup(targetGroupID, categoriesToUpdate, 
sessionID)
```

Inputs:

Name	Description	Type
targetGroupID	The group ID of the group whose category content permissions will be updated.	int
categoriesToUpdate	A List of all categories with their content permissions set.	List<VBCategory>
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the CategoryContentPermissionsUpdateByGroup operation.

Example Client Code:

Note: Login and Logout code is not shown here for the purpose of clarity.

```
//retrieve all categories with group permissions
VBList<VBCategory> allCategoriesWithGroupPermissions =
sll.CategoriesGetAllWithGroupPermissions(targetGroupID, sessionID);
//retrieve default permissions
VBDefaultContentCategoryPermissions defaultPermissions =
sll.ContentCategoryPermissionsDefaultGroupGet(targetGroupID, sessionID);
//retrieve a category
VBCategory targetCategoryForGrantAccessTest =
allCategoriesWithGroupPermissions.Entities.First<VBCategory>();

//loop through all categories and update corresponding permissions
VBCategoryPermission categoryPermissions;
bool isCustomPermissions;
for (int i = 0; i < allCategoriesWithGroupPermissions.Entities.Count; i++)
{
categoryPermissions = new VBCategoryPermission();
if (allCategoriesWithGroupPermissions.Entities[i].Name ==
targetCategoryForGrantAccessTest.Name)
{//found the target category

        //set individual access control permissions for all access
        categoryPermissions.Read = true;
        categoryPermissions.Delete = true;
        categoryPermissions.Edit = true;
        categoryPermissions.Add = true;
        categoryPermissions.Admin = true;
        categoryPermissions.Share = true;
        //determine value of IsCustomized, this is where the default values are
compared
        //to the newly set values to determine if the permissions have been
customized
        isCustomPermissions = ((categoryPermissions.Add != defaultPermissions.Add)
|| (categoryPermissions.Admin != defaultPermissions.Admin) ||
(categoryPermissions.Delete != defaultPermissions.Delete) ||
(categoryPermissions.Edit != defaultPermissions.Edit) || (categoryPermissions.Read
!= defaultPermissions.Read));
        categoryPermissions.IsCustomized = isCustomPermissions;
        //anytime permissions are updated the IsCategoryValueChanged value must be
true
```

```
        allCategoriesWithGroupPermissions.Entities[i].IsCategoryValueChanged =
true;
        allCategoriesWithGroupPermissions.Entities[i].CategoryPermission =
categoryPermissions;
}
else
{//all other categories

        if (allCategoriesWithGroupPermissions.Entities[i].CategoryPermission ==
null)
        {
//no permissions are defined for this category, so just default all permissions to
false
                categoryPermissions.Read = false;
                categoryPermissions.Delete = false;
                categoryPermissions.Edit = false;
                categoryPermissions.Add = false;
                categoryPermissions.Admin = false;
                categoryPermissions.Share = false;
                categoryPermissions.IsCustomized = false;

allCategoriesWithGroupPermissions.Entities[i].IsCategoryValueChanged = false;
        }
        else
        {
                //use the existing permissions for this category
                categoryPermissions.Read =
allCategoriesWithGroupPermissions.Entities[i].CategoryPermission.Read;
                categoryPermissions.Delete =
allCategoriesWithGroupPermissions.Entities[i].CategoryPermission.Delete;
                categoryPermissions.Edit =
allCategoriesWithGroupPermissions.Entities[i].CategoryPermission.Edit;
                categoryPermissions.Add =
allCategoriesWithGroupPermissions.Entities[i].CategoryPermission.Add;
                categoryPermissions.Admin =
allCategoriesWithGroupPermissions.Entities[i].CategoryPermission.Admin;
                categoryPermissions.Share =
allCategoriesWithGroupPermissions.Entities[i].CategoryPermission.Share;
                categoryPermissions.IsCustomized =
allCategoriesWithGroupPermissions.Entities[i].CategoryPermission.IsCustomized;
                allCategoriesWithUserPermissions.Entities[i].IsCategoryValueChanged
= true; //must be set to true anytime permissions differ from the default
permissions

        }

        allCategoriesWithUserPermissions.Entities[i].CategoryPermission =
categoryPermissions;
}
}

//convert vblist to list
List<VBCategory> categoriesToUpdate =
allCategoriesWithGroupPermissions.ToList<VBCategory>();

//make the update
VBVoidData updateCategoryPermissionsResponse =
sll.CategoryContentPermissionsUpdateByGroup(targetGroupID, categoriesToUpdate,
sessionID);
if (updateCategoryPermissionsResponse.Exception != null) //update failed.  Failure
code here
```

Set Group Access Control Level Permissions for Individual Units of Content

The VEMS Mystro API provides the ability to retrieve and update the access control permission levels of individual units of content contained within content categories, at the group level. There are 3 methods involved with the retrieval and update of the access control permissions for individual units of content. Each method will be explained below and then a code sample demonstrating their use will be presented.

Method 1:

Syntax:

```
        ContentGetByCategoryWithGroupPermissions(targetGroupID, targetCategoryID,
sessionID)
```

Inputs:

Name	Description	Type
targetGroupID	The group ID of the group whose individual content permissions will be retrieved.	int
targetCategoryID	The category ID of the category whose content is being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
        VBList<VBAdminContent>
```

The **VBList<VBAdminContent>** object returned is a **VBList** of **VBAdminContent** objects. Each VBAdminContent object represents an individual unit of content. Within each VBAdminContent object there is a `ContentPermission` field which will contain a **VBContentPermission** object, if and only if there are permissions assigned, otherwise this field is null. Also, examine the `Exception` field to determine if an error occurred during the ContentGetByCategoryWithGroupPermissions operation.

Method 2:

Syntax:

```
        ContentCategoryPermissionsDefaultGroupGet(targetGroupID, sessionID)
```

Inputs:

Name	Description	Type
targetGroupID	The group ID of the group whose default category permissions are being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBDefaultContentCategoryPermissions object
```

The **VBDefaultContentCategoryPermissions** object returned contains the default content category permissions assigned to the group. These defaults are determined by unioning all the category permissions of the feature sets that the group belongs too. The `Exception` field should be examined to determine if an error occurred during the ContentCategoryPermissionsDefaultGroupGet operation.

Method 3:

Syntax:

```
        ContentPermissionsUpdateByGroup(targetGroupID, allContentToUpdate,
sessionID)
```

Inputs:

Name	Description	Type
targetGroupID	The group ID of the group whose individual content permissions will be updated.	int
allContentToUpdate	List of content whose permissions will be updated.	List<VBAdminContent>
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentPermissionsUpdateByGroup operation.

Example Client Code:

Note: Login and Logout code is not shown here for the purpose of clarity.

```
//retrieve all content within category
VBList<VBAdminContent> allContentWithinCategory;
allContentWithinCategory =
sll.ContentGetByCategoryWithGroupPermissions(targetGroupID,targetCategoryID,sessio
nID);
//retrieve unit of content to work with
VBAdminContent testContentToUpdate =
allContentWithinCategory.Entities.First<VBAdminContent>();
//retrieve default permissions
VBDefaultContentCategoryPermissions defaultPermissions =
sll.ContentCategoryPermissionsDefaultGroupGet(targetGroupID, sessionID);

//loop through all the retrieved content and update permissions
VBContentPermission contentPermissions;
for (int i = 0; i < allContentWithinCategory.Entities.Count; i++)
{
```

```
//get new category permission object
contentPermissions = new VBContentPermission();
bool isContentPermissionCustomized;
if (allContentWithinCategory.Entities[i].Title == testContentToUpdate.Title)
{//found target content

        //set permission on the target content (grant read and edit access)
        contentPermissions.Admin = false;
        contentPermissions.Default = false;
        contentPermissions.Delete = false;
        contentPermissions.Edit = true;
        contentPermissions.Read = true;
        contentPermissions.Share = true;

        isContentPermissionCustomized = ((contentPermissions.Admin !=
defaultPermissions.Admin) || (contentPermissions.Delete !=
defaultPermissions.Delete) || (contentPermissions.Edit != defaultPermissions.Edit)
|| (contentPermissions.Read != defaultPermissions.Read));
        contentPermissions.IsCustomized = isContentPermissionCustomized;
}
else
{//all other content
        if (allContentWithinCategory.Entities[i].ContentPermission == null)
        {
                //content has no permissions assigned, so default all permissions to
false
                contentPermissions.Admin = false;
                contentPermissions.Default = false; //Default is always false
                contentPermissions.Delete = false;
                contentPermissions.Edit = false;
                contentPermissions.Read = false;
                contentPermissions.Share = false;
                contentPermissions.IsCustomized = false;
        }
        else
        {
                //content has existing permissions, so just use those
                contentPermissions.Admin =
allContentWithinCategory.Entities[i].ContentPermission.Admin;
                contentPermissions.Default =
allContentWithinCategory.Entities[i].ContentPermission.Default; //Default is
always false
                contentPermissions.Delete =
allContentWithinCategory.Entities[i].ContentPermission.Delete;
                contentPermissions.Edit =
allContentWithinCategory.Entities[i].ContentPermission.Edit;
                contentPermissions.Read =
allContentWithinCategory.Entities[i].ContentPermission.Read;
                contentPermissions.Share =
allContentWithinCategory.Entities[i].ContentPermission.Share;
                contentPermissions.IsCustomized =
allContentWithinCategory.Entities[i].ContentPermission.IsCustomized;
        }
}
//assign permissions to current content record
allContentWithinCategory.Entities[i].ContentPermission = contentPermissions;
}

//convert VBList to List
List<VBAdminContent> allContentWithCategoryList =
allContentWithinCategory.ToList<VBAdminContent>();

//make update
VBVoidData updateContentPermissionResponse =
sll.ContentPermissionsUpdateByGroup(targetGroupID, targetedContent, sessionID);

if (updateContentPermissionResponse.Exception != null) //failure code here
```

Set User Access Control Level Permissions for Content Categories

The VEMS Mystro API provides the ability to retrieve and update the access control level permissions for content categories and individual units of content contained therein at the user level. Setting access control level permissions for content categories will be explored

first, followed by setting access control permissions for individual units of content in 20.4. There are 3 methods involved with the retrieval and update of content category access control permissions. Each method will be explained below and then a code sample demonstrating their use will be presented.

Method 1:

Syntax:

```
CategoriesGetAllWithUserPermissions(targetUserID, sessionID)
```

Inputs:

Name	Description	Type
targetUserID	The user ID of the user whose category permissions are being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBCategory>
```

The **VBList<VBCategory>** object returned is a **VBList** of **VBCategory** objects. Within each VBCategory object there is a `CategoryPermission` field which will contain a VBCategoryPermission object, if and only if there are permissions assigned, otherwise this field is null. The `Exception` field should be examined to determine if an error occurred during the CategoriesGetAllWithUserPermissions operation.

Method 2:

Syntax:

```
ContentCategoryPermissionsDefaultUserGet(targetUserID, sessionID)
```

Inputs:

Name	Description	Type
targetUserID	The user ID of the user whose default category permissions are being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBDefaultContentCategoryPermissions object
```

The **VBDefaultContentCategoryPermissions** object returned contains the default content category permissions assigned to the user. These defaults are determined by unioning all the category permissions of the feature sets that the user belongs too. Examine the `Exception` field to determine if an error occurred during the ContentCategoryPermissionsDefaultUserGet operation.

Method 3:

Syntax:

```
CategoryContentPermissionsUpdateByUser(targetUserID, categoriesToUpdate,
sessionID)
```

Inputs:

Name	Description	Type
targetUserID	The user ID of the user whose category content permissions will be updated.	int
categoriesToUpdate	List of all categories with their content permissions set.	List<VBCategory>
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the CategoryContentPermissionsUpdateByGroup operation.

Example Client Code:

Note: Login and Logout code is not shown here for the purpose of clarity.

```
//retrieve all categories with user permissions
VBList<VBCategory> allCategoriesWithUserPermissions =
sll.CategoriesGetAllWithUserPermissions(targetUserID, sessionID);
//retrieve default permissions
VBDefaultContentCategoryPermissions defaultPermissions =
sll.ContentCategoryPermissionsDefaultUserGet(userID, sessionID);
//retrieve a category
VBCategory targetCategoryForGrantAccessTest =
allCategoriesWithUserPermissions.Entities.First<VBCategory>();

//loop through all categories and update corresponding permissions
VBCategoryPermission categoryPermissions;
bool isCustomPermissions;
for (int i = 0; i < allCategoriesWithUserPermissions.Entities.Count; i++)
{
categoryPermissions = new VBCategoryPermission();
if (allCategoriesWithUserPermissions.Entities[i].Name ==
targetCategoryForGrantAccessTest.Name)
{//found the target category

        //set individual access control permissions for all access
        categoryPermissions.Read = true;
        categoryPermissions.Delete = true;
        categoryPermissions.Edit = true;
        categoryPermissions.Add = true;
        categoryPermissions.Admin = true;
        categoryPermissions.Share = true;
        //determine value of IsCustomized
        isCustomPermissions = ((categoryPermissions.Add != defaultPermissions.Add)
|| (categoryPermissions.Admin != defaultPermissions.Admin) ||
(categoryPermissions.Delete != defaultPermissions.Delete) ||
(categoryPermissions.Edit != defaultPermissions.Edit) || (categoryPermissions.Read
!= defaultPermissions.Read));
        categoryPermissions.IsCustomized = isCustomPermissions;
```

```
        allCategoriesWithUserPermissions.Entities[i].IsCategoryValueChanged = true;
        allCategoriesWithUserPermissions.Entities[i].CategoryPermission =
categoryPermissions;
}
else
{//all other categories

        if (allCategoriesWithUserPermissions.Entities[i].CategoryPermission ==
null)
        {
                //no permissions are defined, so just default all permissions to
false
                categoryPermissions.Read = false;
                categoryPermissions.Delete = false;
                categoryPermissions.Edit = false;
                categoryPermissions.Add = false;
                categoryPermissions.Admin = false;
                categoryPermissions.Share = false;
                categoryPermissions.IsCustomized = false;
                allCategoriesWithUserPermissions.Entities[i].IsCategoryValueChanged
= false;
        }
        else
        {
                //use the existing permissions
                categoryPermissions.Read =
allCategoriesWithUserPermissions.Entities[i].CategoryPermission.Read;
                categoryPermissions.Delete =
allCategoriesWithUserPermissions.Entities[i].CategoryPermission.Delete;
                categoryPermissions.Edit =
allCategoriesWithUserPermissions.Entities[i].CategoryPermission.Edit;
                categoryPermissions.Add =
allCategoriesWithUserPermissions.Entities[i].CategoryPermission.Add;
                categoryPermissions.Admin =
allCategoriesWithUserPermissions.Entities[i].CategoryPermission.Admin;
                categoryPermissions.Share =
allCategoriesWithUserPermissions.Entities[i].CategoryPermission.Share;
                categoryPermissions.IsCustomized =
allCategoriesWithUserPermissions.Entities[i].CategoryPermission.IsCustomized;
                allCategoriesWithUserPermissions.Entities[i].IsCategoryValueChanged
= true; //must be set to true anytime permissions differ from the default
permissions
        }

        allCategoriesWithUserPermissions.Entities[i].CategoryPermission =
categoryPermissions;
}
}

//convert vblist to list
List<VBCategory> categoriesToUpdate =
allCategoriesWithUserPermissions.ToList<VBCategory>();

//make the update
VBVoidData updateCategoryPermissionsResponse =
sll.CategoryContentPermissionsUpdateByUser(userID, categoriesToUpdate, sessionID);
```

Set User Access Control Level Permissions for Individual Units of Content

The VEMS Mystro API provides the ability to retrieve and update the access control permission levels of individual units of content contained within content categories, at the user level. There are 3 methods involved with the retrieval and update of the access control permissions for individual units of content. Each method will be explained below and then a code sample demonstrating their use will be presented.

Method 1:

Syntax:

```
ContentGetByCategoryWithUserPermissions(targetUserID, targetCategoryID, sessionID)
```

Inputs:

Name	Description	Type
targetUserID	The user ID of the user whose individual content permissions will be retrieved.	int
targetCategoryID	The category ID of the category whose content is being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBAdminContent>
```

The **VBList<VBAdminContent>** object returned is a **VBList** of **VBAdminContent** objects. Each VBAdminContent object represents an individual unit of content. Within each VBAdminContent object there is a `ContentPermission` field which will contain a VBContentPermission object, if and only if there are permissions assigned, otherwise this field is null. The `Exception` field should be examined to determine if an error occurred during the ContentGetByCategoryWithUserPermissions operation.

Method 2:

Syntax:

```
ContentCategoryPermissionsDefaultUserGet(targetUserID, sessionID)
```

Inputs:

Name	Description	Type
targetUserID	The user ID of the user whose default category permissions are being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBDefaultContentCategoryPermissions object
```

The **VBDefaultContentCategoryPermissions** object returned contains the default content category permissions assigned to the group. These defaults are determined by unioning all the category permissions of the feature sets that the group belongs too. The `Exception` field should be examined to determine if an error occurred during the ContentCategoryPermissionsDefaulUserGet operation.

Method 3:

Syntax:

```
ContentPermissionsUpdateByUser(targetUserID, allContentToUpdate, sessionID)
```

Inputs:

Name	Description	Type
targetUserID	The user ID of the user whose individual content permissions will be updated.	int
allContentToUpdate	List of content whose permissions will be updated.	List<VBAdminContent>
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentPermissionsUpdateByUser operation.

Example Client Code:

Note: Login and Logout code is not shown here for the purpose of clarity.

```
//retrieve all content within category
VBList<VBAdminContent> allContentWithinCategory =
sll.ContentGetByCategoryWithUserPermissions(targetUserID, targetCategoryID,
sessionID);
//retrieve default permissions
VBDefaultContentCategoryPermissions defaultPermissions =
sll.ContentCategoryPermissionsDefaultUserGet(userID, sessionID);

//retrieve unit of content to work with
VBAdminContent testContentToUpdate =
allContentWithinCategory.Entities.First<VBAdminContent>();

//loop through all the retrieved content and update permissions
VBContentPermission contentPermissions;
for (int i = 0; i < allContentWithinCategory.Entities.Count; i++)
{
//get new category permission object
contentPermissions = new VBContentPermission();
bool isContentPermissionCustomized;
if (allContentWithinCategory.Entities[i].Title == testContentToUpdate.Title)
{//found target content

        //set permission on the target content (grant read and edit access)
        contentPermissions.Admin = false;
        contentPermissions.Default = false;
        contentPermissions.Delete = false;
        contentPermissions.Edit = true;
        contentPermissions.Read = true;
        contentPermissions.Share = true;

        isContentPermissionCustomized = ((contentPermissions.Admin !=
defaultPermissions.Admin) || (contentPermissions.Delete !=
defaultPermissions.Delete) || (contentPermissions.Edit != defaultPermissions.Edit)
|| (contentPermissions.Read != defaultPermissions.Read));
        contentPermissions.IsCustomized = isContentPermissionCustomized;
}
```

```
else
{//all other content
        if (allContentWithinCategory.Entities[i].ContentPermission == null)
        {
                //content has no permissions assigned, so default all permissions to
false
                contentPermissions.Admin = false;
                contentPermissions.Default = false; // Default is always false
                contentPermissions.Delete = false;
                contentPermissions.Edit = false;
                contentPermissions.Read = false;
                contentPermissions.Share = false;
                contentPermissions.IsCustomized = false;
        }
        else
        {
                //content has existing permissions, so just use those
                contentPermissions.Admin =
allContentWithinCategory.Entities[i].ContentPermission.Admin;
                contentPermissions.Default =
allContentWithinCategory.Entities[i].ContentPermission.Default; // Default is
always false
                contentPermissions.Delete =
allContentWithinCategory.Entities[i].ContentPermission.Delete;
                contentPermissions.Edit =
allContentWithinCategory.Entities[i].ContentPermission.Edit;
                contentPermissions.Read =
allContentWithinCategory.Entities[i].ContentPermission.Read;
                contentPermissions.Share =
allContentWithinCategory.Entities[i].ContentPermission.Share;
                contentPermissions.IsCustomized =
allContentWithinCategory.Entities[i].ContentPermission.IsCustomized;
        }
}
//assign permissions to current content record
allContentWithinCategory.Entities[i].ContentPermission = contentPermissions;
}

//convert VBList to List
List<VBAdminContent> allContentWithCategoryList =
allContentWithinCategory.ToList<VBAdminContent>();

//make update
VBVoidData updateContentPermissionResponse =
sll.ContentPermissionsUpdateByUser(targetUserID, targetedContent, sessionID);

if (updateContentPermissionResponse.Exception != null) //failure code here
```

Set Content as Public

The VEMS Mystro API provides the ability to designate an individual unit of content as **Public**. This means that all users implicitly receive **View** access to the content.

Syntax:

```
ContentPublicStatusSet(contentID, publicStatus, sessionID)
```

Inputs:

Name	Description	Type
contentID	The content ID of the content whose public status is being set.	int
publicStatus	The value indicating whether or not the content is public.	boolean
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentPublicStatusSet operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful

//Get content of interest from GUI, or search, etc (not shown here)
VBContent contentOfInterest;

//set the content as public
VBVoidData setContentToPublicResponse =
sll.ContentPublicStatusSet(contentOfInterest.ContentID, true,
mySession.SessionID);

if (setContentToPublicResponse.Exception != null)
{
//failure path
}
else
{
//success path
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
//failure path
}
else
{
//success path
}
```

Get Public Status of Content

The VEMS Mystro API provides the ability to retrieve the **Public** status of an individual unit of content.

Syntax:

```
ContentGetPublicStatus(contentID, sessionID)
```

Inputs:

Name	Description	Type
contentID	The content ID of the content whose public status is being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBBoolData object
```

The **VBBoolData** object returned allows client code to determine the **Public** status of a unit of content by examining the `ReturnValue` field of the object. If this field is set to `true`, the content is **Public**, otherwise the content is not designated as Public. The `Exception` field should be examined to determine if an error occurred during the ContentGetPublicStatus operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful

//Get content of interest from GUI, or search, etc (not shown here)
VBContent contentOfInterest;

//retrieve public status
VBBoolData contentOfInterestPublicStatus =
sll.ContentGetPublicStatus(contentOfInterest.ContentID, mySession.SessionID);

bool contentStatus;
if (contentOfInterestPublicStatus.Exception != null)
{
//failure path
}
else
{
//success path
contentStatus = contentOfInterestPublicStatus.ReturnValue;
//take action based on status
}

//log out
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //failure path
```

Set Content as Private

The VEMS Mystro API provides the ability to designate a unit of content as **Private**. This means that the content does not inherit any permissions set at the content category level. Only the individual content permissions set for a user and his group(s) are unioned and applied.

Syntax:

```
ContentPrivateStatusSet(contentID, privateStatus, sessionID)
```

Inputs:

Name	Description	Type
contentID	The content ID of the content whose private status is being set.	int
privateStatus	The value indicating whether or note the content is private.	boolean
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentPrivateStatusSet operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful

//Get content of interest from GUI, or search, etc (not shown here)
VBContent contentOfInterest;

//set private status to true
VBVoidData setPrivateStatusResponse =
sll.ContentPrivateStatusSet(contentOfInterest.ContentID, true,
mySession.SessionID);

if (setPrivateStatusResponse.Exception != null)
{
//failure path
}
else
{
//success path
}

//log out
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //failure path
```

Get Private Status of Content

The VEMS Mystro API provides the ability to retrieve the **Private** status of an individual unit of content.

Syntax:

```
ContentGetPrivateStatus(contentID, sessionID)
```

Inputs:

Name	Description	Type
contentID	The content ID of the content whose private status is being retrieved.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBBoolData object
```

The **VBBoolData** object returned allows client code to determine the **Private** status of a unit of content by examining the `ReturnValue` field of the object. If this field is set to true, the content is **Private**, otherwise the content is not designated as Private. The `Exception` field should be examined to determine if an error occurred during the ContentGetPrivateStatus operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
```

```
//verify that login was successful

//Get content of interest from GUI, or search, etc (not shown here)
VBContent contentOfInterest;

//retrieve private status
VBBoolData contentOfInterestPrivateStatus =
sll.ContentGetPrivateStatus(contentOfInterest.ContentID, mySession.SessionID);

bool contentStatus;
if (contentOfInterestPrivateStatus.Exception != null)
{
//failure path
}
else
{
//success path
contentStatus = contentOfInterestPrivateStatus.ReturnValue;
//take action based on status
}

//log out
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //failure path
```

Set Default Status of Content

The VEMS Mystro API does *not* provide a direct method to designate a unit of content as having **Default** status. A unit of content with Default status means that the access control permissions as discussed in section 20 are in effect. A unit of content is designated as having Default status by setting both the **Public** and **Private** status to **false**.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful
//Get content of interest from GUI, or search, etc (not shown here)


VBContent contentOfInterest;


//set private status to false
VBVoidData setPrivateStatusResponse =
sll.ContentPrivateStatusSet(contentOfInterest.ContentID, false,
mySession.SessionID);
//set public status to false
VBVoidData setPublicStatusResponse =
sll.ContentPublicStatusSet(contentOfInterest.ContentID, false,
mySession.SessionID);
//verify that both statuses were set without errors
if (setPrivateStatusResponse.Exception != null && setPublicStatusResponse.Exception
!= null)
{
//failure path
}
else
{
//success path
}
//log out
```

```
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) //failure path
```

Get Default Status of Content

The VEMS Mystro API does *not* provide a direct method to retrieve the **Default** status of a unit of content. A unit of content with Default status means that the access control permissions as discussed in section 20 are in effect. A unit of content is designated as having Default status when both the **Public** and **Private** status are set to **false**. A developer can write code to retrieve the public and private status of a unit of content and test to determine if they are both false.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
//verify that login was successful
//Get content of interest from GUI, or search, etc (not shown here)
VBContent contentOfInterest;
//retrieve public status
VBBoolData contentOfInterestPublicStatus =
sll.ContentGetPublicStatus(contentOfInterest.ContentID, mySession.SessionID);
//retrieve private status
VBBoolData contentOfInterestPrivateStatus =
sll.ContentGetPrivateStatus(contentOfInterest.ContentID, mySession.SessionID);

if (contentOfInterestPublicStatus.Exception != null &&
contentOfInterestPrivateStatus.Exception != null)
{
//failure path
}
else
{
//success path
bool isContentSetToDefaultStatus;
if (!(contentOfInterestPublicStatus.ReturnValue &&
contentOfInterestPrivateStatus.ReturnValue))
{
        isContentSetToDefaultStatus = true;
}
else
{
isContentSetToDefaultStatus = false;
}
}
//do something now that we know the results
//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
//failure path
}
else
{
//success path
}
```

Chapter 7

Video Content

Topics in this document

Adding Video Content 45
Add a Standalone Video File 45
Add a Stored Entered URL 49
Add a Live Entered URL 53

Adding Video Content

The VEMS Mystro API provides the ability to add video content to the system. There are a couple of different manners to add content to the system. The first manner is to upload a standalone video file. The second manner is to add an entered URL, which can originate from a live source or a stored server. The API methods used for each manner will be examined followed by a comprehensive example for each.

Add a Standalone Video File

The VEMS Mystro API provides the ability to add a standalone video file to the system. There are 5 API calls which have to occur in order for a standalone video file to be successfully ingested into a VEMS Mystro ecosystem.

The first method, **IsFileIngestableBasedOnExtension**, examines the video file and determines if it is compatible with the available stored servers in the ecosystem.

The next method, **ContentSeedCreateForAddVideo**, creates a ContentID for the new video.

Then a content title and description are associated with the new content record with the **ContentTitleDescriptionUpdate** method.

The **ContentUploadedVideoAdd** method starts the video ingestion process.

The method **ContentFtpStatusGetAllByUser** is used to verify that the video was successfully FTP'd. If the file was not successfully FTP'd the video ingestion process will fail.

Method 1:

Syntax:

```
IsFileIngestableBasedOnExtension(fileName, sessionID)
```

Inputs:

Name	Description	Type
fileName	The file name (do not include path info) of the video.	string
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBBoolData object
```

The **VBBoolData** object returned allows client side code to determine if the video is ingestible by examining its `ReturnValue` field. If the value is **true**, the video is ingestible otherwise the video is not ingestible. The `Exception` field should also be examined to determine if any errors occurred during the IsFileIngestableBasedOnExtension operation.

Method 2:

Syntax:

```
ContentSeedCreateForAddVideo(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned allows client side code to retrieve the new content ID by examining its `ReturnValue` field. This content ID will be assigned to the new video. The `Exception` field should also be examined to determine if any errors occurred during the ContentSeedCreateForAddVideo operation.

Method 3:

Syntax:

```
ContentTitleDescriptionUpdate(vbContentObj, sessionID)
```

Inputs:

Name	Description	Type
vbContentObj	The object representing the new video content.	VBContent
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **vbContentObj** prior to passing it to ContentTitleDescriptionUpdate.

Name	Description	Type
ContentID	The content ID returned from ContentSeedCreateforAddVideo	int
Title	The title of the content.	string
Description	The description of the content.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentTitleDescriptionUpdate operation.

Method 4:

Syntax:

```
ContentUploadedVideoAdd(fileName, contentID, sessionID)
```

Inputs:

Name	Description	Type
fileName	The file name, with extension (no path info) of the video file.	string
contentID	The content ID returned from ContentSeedCreateForAddVideo	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentUploadVideoAdd operation.

Method 5:

Syntax:

```
ContentFtpStatusGetAllByUser(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBRequestFtp> object
```

The **VBList<VBRequestFtp>** object returned is a List of **VBRequestFtp** objects. Each VBRequestFtp object corresponds to an FTP requested initiated by the user that invoked the method. There will be a VBRequestFtp object for each video that is added to the system. The `Exception` field should be examined to determine if an error occurred during the ContentFtpStatusGetAllByUser operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null)
{
throw new Exception("UserLogin failed.  See the ExceptionLog table for details");
}

string testVideoFilenameWithExtension; //get filename from GUI, etc (not shown
here).
string completeSourcePathNameOfVideoFile; //get filename from GUI, etc (not shown
here).
//copy the standalone video file into the VOD folder.  This folder structure should
exist on every PC based client.  The API is expecting the video file to be there.
File.Copy(
        completeSourcePathNameOfVideoFile,
        @"c:\inetpub\ftproot\VBrick\VOD\" + testVideoFilenameWithExtension
 );

//Step 1
//determine if the video file is ingestible
VBBoolData isNewVideoIngestible =
sll.IsFileIngestableBasedOnExtension(testVideoFilenameWithExtension,
mySession.SessionID);
if (isNewVideoIngestible.Exception != null)
{
throw new Exception("IsFileIngestableBasedOnExtension failed.  See the ExceptionLog
table for details");
}

if (!isNewVideoIngestible.ReturnValue)
{//the video file type is not compatible with the stored servers in the system
throw new Exception("The video file is not ingestible");
}

//Step 2
//generate a Content ID for the new video
VBIntData contentIDOfNewVideo =
sll.ContentSeedCreateForAddVideo(mySession.SessionID);
if (contentIDOfNewVideo.Exception != null)
{
throw new Exception("ContentSeedCreateForAddVideo failed.  See the ExceptionLog
table for details");
}


string videoTitle; //get from GUI (not shown here)
string videoDescription; //get from GUI (not shown here)

//create a VBContent object representing the new video
VBContent newUploadedVideoContent = new VBContent();
newUploadedVideoContent.ContentID = contentIDOfNewVideo.ReturnValue;
newUploadedVideoContent.Description = videoDescription;
newUploadedVideoContent.Title = videoTitle;

//Step 3
//assign the title and description to the new video content
VBVoidData updateContentTitleAndDescResponse =
sll.ContentTitleDescriptionUpdate(newUploadedVideoContent, mySession.SessionID);
if (updateContentTitleAndDescResponse.Exception != null)
{
throw new Exception("ContentTitleDescriptionUpdate failed.  See the ExceptionLog
table for details");
}

//Step 4
//upload the video file and start ingesting it
VBVoidData uploadVideoResponse =
sll.ContentUploadedVideoAdd(testVideoFilenameWithExtension,
contentIDOfNewVideo.ReturnValue, mySession.SessionID);
if (uploadVideoResponse.Exception != null)
{
throw new Exception("ContentUploadedVideoAdd failed.  See the ExceptionLog table
for details");
```

```
}

//ensure that the video file was successfully FTP'd from the VOD folder
VBList<VBRequestFtp> allUserRecordingsFtpStatus =
sll.ContentFtpStatusGetAllByUser(mySession.SessionID);
if (allUserRecordingsFtpStatus.Exception != null)
{
throw new Exception("ContentFtpStatusGetAllByUser failed.  See the ExceptionLog
table for details");
}

//extract the target FTP status
VBRequestFtp targetFtpStatus =
allRequestFtpRecords.Entities.First<VBRequestFtp>(myFtpRequestRecord =>
myFtpRequestRecord.Title == videoTitle);
//determine if FTPing succeeded
bool isFTPdSuccessfully = (targetFtpStatus.EnumRequestStatusTypeID ==
REQUESTSTATUSTYPE.Succeeded) ? true : false;

if (!isFTPdSuccessfully)
{
//the file was not successfully FTP'd.  This means that the video file was not
ingested into the system.  //There most likely is a problem with the FTP
configuration.
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
throw new Exception("UserLogout failed.  See the ExceptionLog table for details");
}
```

Add a Stored Entered URL

The VEMS Mystro API provides the ability to add a stored entered URL for content originating from a stored server. There are 4 API calls which have to occur *in order* for a stored URL to be accepted into a VEMS Mystro ecosystem.

The first method, **ContentSeedCreateForAddVideo** creates a content ID for the new content.

The second method, **ContentTitleDescriptionUpdate**, associates a title and description with the content.

The third method, **ContentAddExternalURL** actually adds the entered URL to the system.

The last method, **ContentIsSeedUpdate** essentially activates the content and makes it searchable and viewable.

Method 1:

Syntax:

```
ContentSeedCreateForAddVideo(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned allows client side code to retrieve the new content ID by examining its `ReturnValue` field. This content ID will be assigned to the new stored URL. The `Exception` field should also be examined to determine if any errors occurred during the ContentSeedCreateForAddVideo operation.

Method 2:

Syntax:

```
ContentTitleDescriptionUpdate(vbContentObj, sessionID)
```

Inputs:

Name	Description	Type
vbContentObj	The object representing the new video content.	VBContent
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **vbContentObj** prior to passing it to ContentTitleDescriptionUpdate.

Name	Description	Type
ContentID	The content ID returned from ContentSeedCreateforAddVideo	int
Title	The title of the content.	string
Description	The description of the content.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentTitleDescriptionUpdate operation.

Method 3:

Syntax:

```
ContentAddExternalURL(vbContentObj, vbThumbnailObj, sessionID)
```

Inputs:

Name	Description	Type
vbContentObj	The object representing the new video content.	VBContent
vbThumbnailObj	The object representing the Thumbnail associated with the content.	VBThumbnail

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **vbContentObj** prior to passing it to ContentAddExternalURL.

Name	Description	Type
contentID	The content ID returned from ContentSeedCreateforAddVideo	int
ContentInstances	Contains a new content instance representing the entered URL.	List<VBContentInstances>

The following fields must be set on the a new **VBContentInstance** object prior to adding it to the ContentInstances List.

Name	Description	Type
bitRate	The bit rate associated with the stored URL.	int
durationSeconds	The duration of the stored URL content, in seconds.	int
enumContentTypeID	The content type ID for a stored URL (3).	int
enumEncodingTypeID	The encoding type ID for the stored URL.	int
isEnteredURL	Flag indicating whether or not the content is an entered URL.	boolean
URL	The text of the stored URL.	string

Method 4:

Syntax:

```
ContentIsSeedUpdate(vbContentObj, sessionID)
```

Inputs:

Name	Description	Type
vbContentObj	The object representing the content whose seed status needs an update.	VBContent
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **vbContentObj** prior to passing it to ContentIsSeedUpdate.

Output:

Name	Description	Type
ContentID	The content ID of the stored URL.	int
isSeedUpdate	Flag indicating whether or not the content is seed content.	boolean

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentIsSeedUpdate operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null)
{
throw new Exception("UserLogin failed.  See the ExceptionLog table for details");
}
//1. generate content ID
VBIntData contentIDOfNewVideo =
sll.ContentSeedCreateForAddVideo(mySession.SessionID);
if (contentIDOfNewVideo.Exception != null)
{
throw new Exception("ContentSeedCreateForAddVideo failed. See ExceptionLog table
for details");
}
string videoTitle; //retrieve from GUI, etc (not shown here)
string videoDescription; //retrieve from GUI, etc (not shown here)

//construct content object
VBContent newContent = new VBContent();
newContent.ContentID = contentIDOfNewVideo.ReturnValue;
newContent.Title = videoTitle;
newContent.Description = videoDescription;

//2. Update title and description
VBVoidData UpdateTitleDescResponse = sll.ContentTitleDescriptionUpdate(newContent,
mySession.SessionID);
if (UpdateTitleDescResponse.Exception != null)
{
throw new Exception("ContentTitleDescriptionUpdate failed. See ExceptionLog table
for details");
}
//populate these stored url parameters from the GUI, etc (not shown here)
int bitRate;
int durationInSeconds;
int encodingTypeID;
string videoURL;

//build up content object
VBContent myNewContent = new VBContent();
myNewContent.ContentID = contentIDOfNewVideo.ReturnValue;
myNewContent.IsSeedContent = false; //must be false to activate the content
//build up a VBContentInstance object
VBContentInstance myContentInstance = new VBContentInstance();
myContentInstance.BitRate = bitRate;
myContentInstance.DurationSeconds = durationInSeconds;
myContentInstance.EnumEncodingTypeID = encodingTypeID;
myContentInstance.IsEnteredURL = true;
myContentInstance.EnumContentTypeID = (int)CONTENTTYPE.Stored;
myContentInstance.URL = videoURL;
//associate the content instance to the content
myNewContent.ContentInstances = new List<VBContentInstance>();
```

```
myNewContent.ContentInstances.Add(myContentInstance);

//pass a null VBThumbnail object (it's not required)
VBThumbnail myContentThumbnail = null;

//3. Add the external URL
VBVoidData addExternalURLResponse = sll.ContentAddExternalURL(myNewContent,
myContentThumbnail, mySession.SessionID);
if (addExternalURLResponse.Exception != null)
{
throw new Exception("ContentAddExternalURL failed. See ExceptionLog table for
details");
}
//4. Update the content's seed status
VBVoidData IsSeedUpdateResponse = sll.ContentIsSeedUpdate(myNewContent,
mySession.SessionID);
if (IsSeedUpdateResponse.Exception != null)
{
throw new Exception("ContentIsSeedUpdate failed.  See ExceptionLog table for
details");
}
//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
throw new Exception("UserLogout failed.  See ExceptionLog table for details");
}
```

Add a Live Entered URL

The VEMS Mystro API provides the ability to add a live entered URL. Like adding a stored URL, there are 4 API calls which have to occur *in order* for a live entered URL to be accepted into a VEMS Mystro ecosystem.

The first method, **ContentSeedCreateForAddVideo** creates a content ID for the new content.

The second method, **ContentTitleDescriptionUpdate**, associates a title and description with the content.

The third method, **ContentAddExternalURL** actually adds the entered URL to the system.

The last method, **ContentIsSeedUpdate** essentially activates the content and makes it searchable and viewable.

Method 1:

Syntax:

```
ContentSeedCreateForAddVideo(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned allows client side code to retrieve the new content ID by examining its `ReturnValue` field. This content ID will be assigned to the new stored URL.

The `Exception` field should also be examined to determine if any errors occurred during the ContentSeedCreateForAddVideo operation.

Method 2:

Syntax:

```
ContentTitleDescriptionUpdate(vbContentObj, sessionID)
```

Inputs:

Name	Description	Type
vbContentObj	The object representing the new video content.	VBContent
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **vbContentObj** prior to passing it to ContentTitleDescriptionUpdate.

Name	Description	Type
ContentID	The content ID returned from ContentSeedCreateforAddVideo	int
Title	The title of the content.	string
Description	The description of the content.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentTitleDescriptionUpdate operation.

Method 3:

Syntax:

```
ContentAddExternalURL(vbContentObj, vbThumbnailObj, sessionID)
```

Inputs:

Name	Description	Type
vbContentObj	The object representing the Live URL content.	VBContent
vbThumbnailObj	The object representing the Thumbnail associated with the content.	VBThumbnail
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **vbContentObj** prior to passing it to ContentAddExternalURL.

Name	Description	Type
contentID	The content ID returned from ContentSeedCreateforAddVideo	int
ContentInstances	Contains a new content instance representing the entered URL.	List<VBContentInstances>

The following fields must be set on the a new **VBContentInstance** object prior to adding it to the ContentInstances List.

Name	Description	Type
bitRate	The bit rate associated with the live URL.	int
durationSeconds	The duration of the live URL content, in seconds.	int
enumContentTypeID	The content type ID for a live URL (3).	int
enumEncodingTypeID	The encoding type ID for the live URL.	int
isEnteredURL	Flag indicating whether or not the content is an entered URL.	boolean
URL	The text of the live URL.	string
isMulticast	Flag indicating whether or not the stream is multicast.	boolean
ZoneIPAddress	The IP Address of the live URL.	string

Method 4:

Syntax:

```
ContentIsSeedUpdate(vbContentObj, sessionID)
```

Inputs:

Name	Description	Type
vbContentObj	The object representing the content whose seed status needs an update.	VBContent
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **vbContentObj** prior to passing it to ContentIsSeedUpdate.

Name	Description	Type
ContentID	The content ID of the live URL.	int
isSeedUpdate	Flag indicating whether or not the content is seed content.	boolean

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentIsSeedUpdate operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null)
{
throw new Exception("UserLogin failed.  See the ExceptionLog table for details");
}

//1. generate content ID
VBIntData contentIDOfNewVideo =
sll.ContentSeedCreateForAddVideo(mySession.SessionID);
if (contentIDOfNewVideo.Exception != null)
{
throw new Exception("ContentSeedCreateForAddVideo failed. See ExceptionLog table
for details");
}

string videoTitle; //retrieve from GUI, etc (not shown here)
string videoDescription; //retrieve from GUI, etc (not shown here)

VBContent newUploadedVideoContent = new VBContent();
newUploadedVideoContent.ContentID = contentIDOfNewVideo.ReturnValue;
newUploadedVideoContent.Description = videoDescription;
newUploadedVideoContent.Title = videoTitle;

//2. Update title and description
VBVoidData UpdateTitleDescResponse;
UpdateTitleDescResponse =
sll.ContentTitleDescriptionUpdate(newUploadedVideoContent, mySession.SessionID);
if (UpdateTitleDescResponse.Exception != null)
{
throw new Exception("ContentTitleDescriptionUpdate failed. See ExceptionLog table
for details");
}

//populate these stored url parameters from the GUI, etc (not shown here)
int bitRate;
int durationInSeconds;
int encodingTypeID;
string videoURL;
string ipAddress;
string isMulticastURL;

//construct VBContent obj
VBContent newExternalLiveUrlContent = new VBContent();
newExternalLiveUrlContent.ContentID = contentIDOfNewVideo.ReturnValue;
newExternalLiveUrlContent.ContentInstances = new List<VBContentInstance>();
//construct VBContentInstance obj
VBContentInstance newExternalLiveUrlContentInstance = new VBContentInstance();
newExternalLiveUrlContentInstance.ZoneIPAddress = ipAddress;
newExternalLiveUrlContentInstance.URL = videoURL;
newExternalLiveUrlContentInstance.IsEnteredURL = true;
newExternalLiveUrlContentInstance.IsMulticast = isMulticastURL;
newExternalLiveUrlContentInstance.EnumContentTypeID = (int)CONTENTTYPE.Live;
newExternalLiveUrlContentInstance.EnumEncodingTypeID = encodingTypeID;
newExternalLiveUrlContentInstance.BitRate = bitRate;
//add instance to content
newExternalLiveUrlContent.ContentInstances.Add(newExternalLiveUrlContentInstance);
//pass a null VBThumbnail object (it's not required)
VBThumbnail myContentThumbnail = null;
```

```
//3. Add the external URL
VBVoidData addExternalURLResponse =
sll.ContentAddExternalURL(newExternalLiveUrlContent, myContentThumbnail,
mySession.SessionID);
if (addExternalURLResponse.Exception != null)
{
throw new Exception("ContentAddExternalURL failed. See ExceptionLog table for
details");
}
//build content obj for seed update
VBContent newExternalUrlContent = new VBContent();
newExternalUrlContent.ContentID = contentIDOfNewVideo.ReturnValue;
newExternalUrlContent.IsSeedContent = false;

//4. Update the content's seed status (activate the content)
VBVoidData IsSeedUpdateResponse = sll.ContentIsSeedUpdate(newExternalUrlContent,
mySession.SessionID);
if (IsSeedUpdateResponse.Exception != null)
{
throw new Exception("ContentIsSeedUpdate failed.  See ExceptionLog table for
details");
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
throw new Exception("UserLogout failed.  See ExceptionLog table for details");
}
```

Chapter 8

Search

Topics in this document

Searching for Content 59
Clear the Content Cache 59
Search for Content. 60
Search for Content Instances. 63

Searching for Content

The VEMS Mystro API provides the ability to search a VEMS ecosystem for content. All types of content are searchable, whether it be content from stored servers, stored or live entered URLs, live broadcasts, live webcasts, and so forth. There are 3 methods used to search for content.

The **SearchContent** and **SearchContentNoCache** methods are the workhorses used to search for and return content. The difference between the two is that SearchContent has slightly better performance because it utilizes the content cache, whereas SearchContentNoCache does not use the content cache.

The last method involved with searching is the **ContentSearchCacheClear** method. This method flushes the content cache. The combination of calling ContentSearchCacheClear and then SearchContent has the same effect as calling SearchContentNoCache.

Search Caveats:

1. The SearchContent and SearchContentNoCache methods automatically apply filtering according to the *Access Control Rights* as well as filtering based on zones. This means that a search will not return any content that a user does not have permission for, or any search results that are not available to his zone.
2. There is no direct to way to arbitrarily retrieve all content available to a specific user. One way to accomplish this task would be to login with the target user, perform a search for all stored content, perform a search for all live content, and then combine the results.
3. There is no direct way to arbitrarily retrieve all content available to a specific group. One way to accomplish this task would be to create a user and assign him to the target group. Do not assign any user level permissions to this "shell" user. Login in with this user and perform a search for all stored content, perform a search for all live content, and then combine the results.

Clear the Content Cache

The VEMS Mystro API provides the ability to deterministically flush the content cache. This method is discussed prior to the methods that implement searching because many times (if not all the time) it is necessary to have up to date search results for whatever processing a developer has in mind. The content cache can become temporarily stale while adding,

deleting and updating content. This could lead to unrepeatable reads and/or phantom reads occurring while searching and processing content. It is recommended that the content cache be cleared directly before calling **SearchContent**. The **SearchContentNoCache** method does not use the content cache so there is no risk of returning stale content data and there is no need to explicitly clear the content cache prior to calling this method.

Syntax:

```
ContentSearchCacheClear(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentSearchCacheClear operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null)
{
throw new Exception("UserLogin failed.  See the ExceptionLog table for details");
}

//clear the cache
VBVoidData clearCacheResponse = sll.ContentSearchCacheClear(mySession.SessionID);
if (clearCacheResponse.Exception != null) throw new
Exception("ContentSearchCacheClear failed");

//now proceed to do search and process content results
//…….if using SearchContent, clear the cache every time before a search is
performed, else use
//SearchContentNoCache

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed.");
```

Search for Content

The **SearchContent** and **SearchContentNoCache** methods are used to perform general content searches.

Syntax:

```
SearchContent(vbSearchFilterObj, sessionID)
SearchContentNoCache(vbSearchFilterObj, sessionID)
```

Inputs:

Name	Description	Type
vbSearchFilterObj	The object representing the search criteria.	VBSearchFilter

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

The most often used fields that can be set on **vbSearchFilterObj** prior to passing it to SearchContent are:

Name	Description	Type
EnumContentTypeID	Value indicating the content type to search for.	int
EnumSearchFieldID	Value indicating the field the search string will be searched against (title, description, etc.).	int
EnumSortByField	Value indicating the field the returned content will be sorted by (title, description, etc.).	int
SearchString	The text to search for.	string
ContentIDs	The ContentIDs to search for. This is only used when searching for content instances.	List<int>
Categories	The search will only take place within the categories added to this list. Leave null to search all categories.	List<VBCategory>
DisplayNum	Number of results to display for pagination purposes.	int
DisplaySection	Section of content that should be displayed (always set to 1).	int
IsSortAsc	Flag indicating whether or not to sort results ascending.	boolean
IncludeInstances	Flag indicating whether or not to include content instance records with content records.	boolean
IsInChannelGuideMode	Flag indicating whether or not to only include content that should be included in the channel guide.	boolean

Name	Description	Type
PresentationOnly	Flag indicating whether or not to only include Presentations, usually originating from Live webcasts.	boolean
ExcludeEnteredURL	Flag indicating whether or not to exclude entered URLS from results.	boolean
ExcludeFlashContent	Flag indicating whether or not to exclude Flash content from results.	boolean
ExcludePresentationContent	Flag indicating whether or not to exclude presentation content from results.	boolean
ExcludeUnviewedContent	Flag indicating whether or not to exclude content that has not yet been viewed.	boolean
ExcludeVODMultiCast	Flag indicating whether or not to exclude multicast content.	boolean
ExcludePendingScheduleEventsIfInChannelGuideMode	Flag indicating whether or not to exclude any pending scheduled events while in Channel Guide mode.	boolean
IncludeOnlyPublishingPointsThatAllowScheduledMultiCast	Flag indicating whether or not to include content that comes from a VOD server that allows scheduled multicast.	boolean

Output:

```
VBContentList<VBContent>
```

The **VBContentList<VBContent>** object returned contains a list of VBContent objects representing all search criteria matches. Note that the VBContentList class is derived from the **VBList** class and either one can be used to store the returned results. The `Exception` field should be examined to determine if any errors occurred during the SearchContent or SearchContentNoCache operations.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");
```

```
//retrieve search string, content type and category IDs (optionally) from GUI (not
shown here)
string searchText;
int contentTypeID;
List<int> onlySearchInTheseCategories;

//create search criteria filter
VBSearchFilter filterForContentSearch = new VBSearchFilter();
//These are the most important fields when performing a search
filterForContentSearch.EnumContentTypeID = contentTypeID; //CONTENTTYPE enum value
(live or stored)
filterForContentSearch.EnumSearchFieldID = (int)SEARCHFIELD.All; //SEARCHFIELD
enum value (title,description,all)
filterForContentSearch.SearchString = searchText; //can be blank if the user does
not want to restrict the returned results to any particular search string.  This
would be blank if retrieving all stored or live content

//can restrict search to limited number of categories if desired, otherwise all
categories are searched
filterForContentSearch.Categories = onlySearchInTheseCategories;

//these fields are used by the GUI and won't affect the results returned from the
search
filterForContentSearch.EnumSortByFieldID = (int)SORTBYFIELD.Title; //SORTBYFIELD
enum value
filterForContentSearch.IsSortAsc = true;
filterForContentSearch.DisplayNum = 10;    //good default value to use
filterForContentSearch.DisplaySection = 1; //good default value to use

//always include content instances unless there is a compelling reason not to
filterForContentSearch.IncludeInstances = true;

//normally don't exclude any content unless there is a compelling reason to do so
filterForContentSearch.IsInChannelGuideMode = false;
filterForContentSearch.PresentationOnly = false;
filterForContentSearch.ExcludeEnteredURL = false;
filterForContentSearch.ExcludeFlashContent = false;
filterForContentSearch.ExcludePresentationContent = false;
filterForContentSearch.ExcludeUnviewedContent = false;
filterForContentSearch.ExcludeVODMulticast = false;
filterForContentSearch.ExcludePendingScheduledEventsIfInChannelGuideMode = false;
filterForContentSearch.IncludeOnlyPublishingPointsThatAllowScheduledMulticast =
false;

//now that the filter has been created, call either SearchContent or
SearchContentNoCache
VBContentList<VBContent> allMatchedContent;
allMatchedContent = sll.SearchContentNoCache(filterForContentSearch,
mySession.SessionID);

if (allMatchedContent.Exception != null) throw new Exception("SearchContentNoCache
failed.");

//process content results here

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed.");
```

Search for Content Instances

The VEMS Mystro API provides the ability to search for content instances with the **SearchContent** and **SearchContentNoCache** methods. The key to searching for content instances with these 2 methods is how the VBSearchFilter object is constructed. The

configuration of the VBSearchFilter object necessary for retrieving content instances is discussed below.

Syntax:

```
VBSearchFilter myNewSearchFilter = new VBSearchFilter();
```

Inputs:

Name	Description	Type
ContentIDs	The content IDs of the content whose instances are being searched for.	List<int>
IncludeInstances	Flag indicating whether or not to include content instances. This must be set to true.	boolean

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve contentIDs of target content from GUI, etc (not shown here)
List<int> allTargetContentIDs;

VBSearchFilter myNewSearchFilter = new VBSearchFilter();
myNewSearchFilter.IncludeInstances = true;
mySearchFilter.ContentIDs = allTargetContentIDs;

VBContentList<VBContent> allFoundContentInstances;
allFoundContentInstances = sll.SearchContent(myNewSearchFilter,
mySession.SessionID);

if (allFoundContentInstances.Exception != null) throw new Exception("SearchContent
failed");

//SearchContent returns VBContent objects.  Within each VBContent object the
ContentInstances
//field contains all the content instances for that content record
foreach(VBContent currentContent in allFoundContentInstances.Entities)
{
foreach(VBContentInstance currentContentInstance in
currentContent.ContentInstances)
{
//access and process content instances here
}
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed.");
```

Chapter 9

Content Instances

Topics in this document

Managing Content Instances 65
Add Content Instance 65
Edit Content Instance 67
Delete Content Instance 68

Managing Content Instances

The VEMS Mystro API provides the ability to add, edit and delete content instances. A unit of content can be associated with different versions of itself; each version can have its own bit rate, content type, encoding type and URL. These different versions are referred to as content instances.

Add Content Instance

The VEMS Mystro API provides the ability to add entered URL content instances to a unit of content.

Syntax:

```
EnteredURLInstanceAdd(contentInstanceObj, sessionID)
```

Inputs:

Name	Description	Type
contentInstanceObj	The object representing the content instance.	VBContentInstance
sessionID	The session ID of the user invoking the method.	string

The following fields must be set on the **contentInstanceObj** prior to passing it to EnteredURLInstanceAdd:

Name	Description	Type
BitRate	The bit rate of the content instance.	int
ContentID	The content ID of the content that this content instance belongs to.	int
ContentInstanceID	The ID of the content instance. This is always set to zero.	int

Name	Description	Type
DurationSeconds	The duration of the content instance, in seconds.	int
EnumContentTypeID	The content tpe of the content instance (live or stored)	int
EnumEncodingTypeID	The encoding type of the content instance.	int
IsEnteredURL	Flag indicating whether or not the content instance the content instance is an entered URL. This is always set to true.	boolean
URL	The URL of the content instance.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the EnteredURLInstanceAdd operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content ID of target content from GUI, etc (not shown here)
int targetContentID;

//retrieve the information needed for the new content instance from GUI, etc (not
shown here)
int bitrate;
int duration;
int contentTypeID;
int encodingTypeID;
string contentInstanceURL;

//construct the content instance object
VBContentInstance myNewContentInstance = new VBContentInstance();
myNewContentInstance.BitRate = bitrate;
myNewContentInstance.ContentID = targetContentID;
myNewContentInstance.ContentInstanceID = 0; //this is always set to zero
myNewContentInstance.DurationSeconds = duration;
myNewContentInstance.EnumContentTypeID = contentTypeID; //value of CONTENTTYPE Enum
myNewContentInstance.EnumEncodingTypeID = encodingTypeID; //value of ENCODINGTYPE
Enum
myNewContentInstance.IsEnteredURL = true; //this is always set to true
myNewContentInstance.URL = contentInstanceURL;

VBVoidData addContentInstanceResponse;
addContentInstanceResponse = sll.EnteredURLInstanceAdd(myNewContentInstance,
mySession.SessionID);

if (addContentInstanceResponse.Exception != null)
{
throw new Exception("EnteredURLInstanceAdd  failed.")
}

//search for the new content instance, etc
```

```
//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null)
{
throw new Exception("UserLogout failed.");
}
```

Edit Content Instance

The VEMS Mystro API provides the ability to edit content instances. Most properties of a content instance can be updated. However, a content instance's content *type* (live or stored) cannot be changed. To accomplish a content type change, the original content instance would have to be deleted instead and then re-added with the desired content type.

Syntax:

```
EnteredURLInstanceUpdate(contentInstanceToEditObj, sessionID)
```

Inputs:

Name	Description	Type
contentInstanceToEditObj	The object representing the content instance to edit.	VBContentInstance
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the EnteredURLInstanceUpdate operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve the target content instance (not shown here)
VBContentInstance instanceOfInterest;

//retrieve new bit rate, duration and URL from GUI,etc (not shown here)
int newBitRate;
int newDuration;
string newURL;

//update fields on the target content instance
instanceOfInterest.BitRate = newBitRate;
instanceOfInterest.DurationSeconds = newDuration;
instanceOfInterest.URL = newURL;

VBVoidData updateInstanceResponse;
updateInstanceResponse = sll.EnteredURLInstanceUpdate(instanceOfInterest,
mySession.SessionID);

if (updateInstanceResponse.Exception != null)
{
throw new Exception("EnteredURLInstanceUpdate failed. See ExceptionLog for
details");
}
```

```
//continue working…
//logout
VBSession logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
```

Delete Content Instance

The VEMS Mystro API provides the ability to delete a content instance. If a unit of content only contains one content instance and that one content instance is being deleted, then the entire unit of content should be deleted as well. This is taken care of with the **deleteContent** parameter.

Syntax:

```
        ContentStoredDeleteFromStoredServer(targetContentInstanceID, deleteContent,
sessionID)
```

Inputs:

Name	Description	Type
targetContentInstanceID	The ID of the content instance to delete.	int
deleteContent	Flag indicating whether or note to remove the entire unit of content or just remove one instance of it.	boolean
sessionID	The session ID of the user invoking the method.	string

Output:

```
        VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentStoredDeleteFromStoredServer operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content ID of content with instance that needs to be removed (not shown
here)
int targetContentID;

VBSearchFilter myNewSearchFilter = new VBSearchFilter();
myNewSearchFilter.IncludeInstances = true; //this must always be set to true
myNewSearchFilter.ContentIDs = new List<int>();
myNewSearchFilter.ContentIDs.Add(targetContentID);

VBContentList<VBContent> allFoundContentInstances;
allFoundContentInstances = sll.SearchContent(myNewSearchFilter,
mySession.SessionID);

if (allFoundContentInstances.Exception != null) throw new Exception("SearchContent
failed");

VBVoidData removeInstanceResponse;
if (allFoundContentInstances.Entities.Count == 1) //only one unit of content should
```

```
be returned
{
VBContent contentToWorkWith = allFoundContentInstances.Entities[0];

//let’s just remove the first instance, and if it’s the only instance remove the
content
int instanceCount = contentToWorkWith.ContentInstances.Count;
if (instanceCount == 1)
{
removeInstanceResponse =
sll.ContentStoredDeleteFromStoredServer(contentToWorkWith.ContentInstances[0].Cont
entInstanceID, true, mySession.SessionID);

if (removeInstanceResponse.Exception != null)
{
throw new Exception ("ContentStoredDeleteFromStoredServer failed")
}
}
else
{
removeInstanceResponse =
sll.ContentStoredDeleteFromStoredServer(contentToWorkWith.ContentInstances[0].Cont
entInstanceID, false, mySession.SessionID);

if (removeInstanceResponse.Exception != null)
{
throw new Exception ("ContentStoredDeleteFromStoredServer failed")
}
}
//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logoutResponse.Exception != null) throw new Exception("UserLogout failed");
}
else
{
//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logoutResponse.Exception != null) throw new Exception("UserLogout failed");
}
```

Chapter 10

Play and Stop Content

Topics in this document

Playing and Stopping Content 71
Play Content. 71
Play Content Instance 73
Stop Playing Content 75
Play Content to Set Top Box (STB) 76
Rebroadcast Content to STB 76
Live Broadcast Content to STB 84
Tune STB to Existing Stream 93

Playing and Stopping Content

The VEMS Mystro API provides the ability to start playing and stop playing any type of content, including broadcasts (live or stored). There is a method for playing content, regardless of content instance as well as another method for playing a specific instance of content. The methods needed to play content will be explored first, followed by the method needed to stop playing content.

Play Content

The VEMS Mystro API provides the ability to play content. All content, including broadcasts (live or stored) are played using the same API methods. There are 2 methods responsible for playing content.

One method called **PlayContentLoadWithPlayerController** is responsible for retrieving a content instance to play along with a **PlayerController** object that contains data needed to control the client side player. The developer cannot control which content instance is returned; the system searches for all content instances of the target content and uses the first one returned, based on search logic. The search logic consists of applying content approval filtering, zones filtering as well as access control filtering.

The other method, **PlayContentLogStart** is responsible for playing a content instance and making a log entry of the play request. Each method will be explained below followed by an example demonstrating their use.

Method 1:

Syntax:

```
PlayContentLoadWithPlayerController(targetContentID, sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the content to play.	int

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBPlayContentInfo object
```

The **VBPlayContentInfo** object returned contains a VBContentInstance object with a URL to play the content along with a **PlayerController** object that contains the JavaScript code used to control the client side player. The `Exception` field should be examined to determine if an error occurred during the PlayContentLoadWithPlayerController operation.

Method 2:

Syntax:

```
PlayContentLogStart(contentInstanceToPlay, sessionID)
```

Inputs:

Name	Description	Type
contentInstanceToPlay	The object representing the content instance to play.	VBContentInstance
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBContentInstance object
```

The **VBContentInstance** object returned contains an ActivityLogID which corresponds to the log entry made in the database for the play request. This ActivityLogID will become useful when it comes time to stop playing the content. The `Exception` field should be examined to determine if an error occurred during the PlayContentLogStart operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content ID of content to play by doing a search based on input from GUI
(not shown)
int targetContentID;

//retrieve playback info
VBPlayContentInfo targetContentPlayInfo;
targetContentPlayInfo = sll.PlayContentLoadWithPlayerController(targetContentID,
mySession.SessionID);

if (targetContentPlayInfo.Exception != null)
{
throw new Exception("PlayContentLoadWithPlayerController failed");
}
```

```
//start playing the content
VBContentInstance playbackResponse;
playbackResponse = sll.PlayContentLogStart(targetContentPlayInfo.ContentInstance,
mySession.SessionID);

//let the video play

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
}
```

Play Content Instance

The VEMS Mystro API provides the ability to playback a specific content instance. This gives the developer control over the content instance that is chosen for playback. Instead of relying on search logic to retrieve a content instance, the developer can retrieve all content instances available for a unit of content and choose a particular one for playback. There are 2 methods responsible for playing content instances. The method used to play a specific content instance is called **PlayContentInstanceLoadWithPlayerController**. This method works almost exactly the same as **PlayContentLoadWithPlayerController** except that it plays the content instance passed to it and doesn't automatically choose one based on search logic. The other method is the same used one to play content (without a specific instance in mind), **PlayContentLogStart**. Each method will be examined below followed by an example.

Method 1:

Syntax:

```
        PlayContentInstanceLoadWithPlayerController(contentInstanceToPlayObj,
sessionID)
```

Inputs:

Name	Description	Type
contentInstanceToPlayObj	The object representing the content instance to play.	VBContentInstance
sessionID	The session ID of the user invoking the method.	string

Output:

```
        VBPlayContentInfo object
```

The **VBPlayContentInfo** object returned contains a VBContentInstance object with a URL to play the content along with a PlayerController object that contains the JavaScript code used to control the client side player. The `Exception` field should be examined to determine if an error occurred during the PlayContentInstanceLoadWithPlayerController operation.

Method 2:

Syntax:

```
        PlayContentLogStart(contentInstanceToPlay, sessionID)
```

Inputs:

Name	Description	Type
contentInstanceToPlay	The object representing the content instance to play.	VBContentInstance
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBContentInstance object
```

The **VBContentInstance** object returned contains an ActivityLogID which corresponds to the log entry made in the database for the play request. This ActivityLogID will become useful when it comes time to stop playing the content. The **`Exception`** field should be examined to determine if an error occurred during the PlayContentLogStart operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content ID of content to play by doing a search based on input from GUI
(not shown)
int targetContentID;

//retrieve all content instances for this unit of content
VBSearchFilter myNewSearchFilter = new VBSearchFilter();
myNewSearchFilter.IncludeInstances = true;
mySearchFilter.ContentIDs = new List<int>();
mySearchFilter.ContentIDs.Add(targetContentID);

VBList<VBContent> allInstancesOfTargetContent;
allInstancesOfTargetContent = sll.SearchContent(myNewSearchFilter,
mySession.SessionID);

if (allInstancesOfTargetContent.Exception != null)
{
throw new Exception("SearchContent failed");
}

//retrieve search criteria from GUI for the desired content instance, let's say a
specific //encoding type (not shown here)
int targetEncodingTypeID;

VBContentInstance targetInstanceToPlay;
targetInstanceToPlay = allInstancesOfTargetContent.Entities.Find(myInstance =>
myInstance.EnumEncodingTypeID == targetEncodingTypeID);

if (targetInstanceToPlay == null) throw new Exception("Could not find target
content instance");

VBPlayContentInfo targetInstancePlayInfo;
targetInstancePlayInfo =
sll.PlayContentInstanceLoadWithPlayerController(targetInstanceToPlay,
mySession.SessionID);

if (targetInstancePlayInfo.Exception != null) throw new Exception("Could not find
playback info for target content instance");

VBContentInstance targetInstanceRet;
targetInstanceRet = sll.PlayContentLogStart(targetInstanceToPlay.ContentInstance,
mySession.SessionID);
```

```
if (targetInstanceRet.Exception != null) throw new Exception("Playback failed");

//let the video play…

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
}
```

Stop Playing Content

The VEMS Mystro API provides the ability to stop playing content. To stop playing content, the value of the **ActivityLogID** field in the VBContentInstance object returned from PlayContentLogStart is needed.

Syntax:

```
PlayContentStop(targetActivityLogID, sessionID)
```

Inputs:

Name	Description	Type
targetActivityLogID	The value of the ActivityLogID field of the VBContentInstance object returned from the PlayContentLogStart method.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the PlayContentStop operation by examining the **`Exception`** field.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content ID of content to play by doing a search based on input from GUI
(not shown)
int targetContentID;

VBPlayContentInfo contentPlayInfo;
contentPlayInfo = sll.PlayContentLoadWithPlayerController(targetContentID,
mySession.SessionID);

if (contentPlayInfo.Exception != null) throw new Exception("Could not find playback
info for target content");

VBContentInstance playContentResponse;
playContentResponse = sll.PlayContentLogStart(contentPlayInfo.ContentInstance,
mySession.SessionID);

if (playContentResponse.Exception != null) throw new Exception("Playback failed");

//let the video play…
```

```
//now let's stop playing the video
VBVoidData stopPlayingVideoResponse =
sll.PlayContentStop(playContentResponse.ActivityLogID, mySession.SessionID);

if (stopPlayingVideoResponse.Exception != null) throw new Exception("Could not stop
playing video");

VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
```

Play Content to Set Top Box (STB)

The VEMS Mystro API provides the ability to play specific content to specific Set Top Boxes (STBs). Playing content to a STB requires setting up a scheduled event. There are 3 types of scheduled events that can be used to play content to a STB. These scheduled events consist of a content rebroadcast event, live broadcast event, and a tuned STB to an existing streamed event. Each type of scheduled event will be explored.

The methods involved with creating each type of scheduled event will be followed by a comprehensive code sample.

Rebroadcast Content to STB

The VEMS Mystro API provides the ability to rebroadcast Stored content to a STB. The methods used to create a Rebroadcast Content scheduled event and send it to a STB will be examined followed by a comprehensive code example.

Method 1:

Syntax:

```
        DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
supportsDaylightSvgsTime, daylightsSavingsTransitionStartMonth,
daylightSavingsTransitionEndMonth, sessionID)
```

Inputs:

Name	Description	Type
utcOffsetInMin	The UTC offset (in minutes) of the target timezone.	int
supportsDaylightSvgsTime	Flag that indicates if the target time zone supports daylight savings time.	boolean
daylightSavingsTransitionStartMonth	The month when daylight savings time begins in the target time zone (January = 1, etc.)	int
daylightSavingsTimeTransitionEndMonth	The month when daylight savings time ends in the target time zone (January = 1, etc.)	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
        VBIntData object
```

The **VBIntData** object returned contains the timeZoneID of the target time zone in its `ReturnValue` field. This value is needed when setting up a scheduled event. The `Exception` field can also be examined to determine if an error occurred during the DateTimeGetBestGuessEnumTimeZoneID operation.

Method 2:

Syntax:

```
ScheduleEventContentInfoGet(targetContentID, sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the content to rebroadcast.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBScheduledEventContentInfo object
```

The **VBScheduledEventContentInfo** object returned contains information about the content to rebroadcast that the scheduling module needs. The `Exception` field should be examined to determine if an error occurred during the ScheduleEventContentInfoGet operation.

Method 3:

Syntax:

```
ScheduleEventContentSeedCreateFromContentWithMetadata(targetContentID,
sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the content to record.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned contains the newly generated content ID for the rebroadcast content event in its ReturnValue field. The `Exception` field should be examined to determine if an error occurred during the ScheduleEventContentSeedCreateFromContentWithMetadata operation.

Method 4:

Syntax:

```
ContentTitleDescriptionUpdate(targetContentObj, sessionID)
```

Inputs:

Name	Description	Type
targetContentObj	The object representing the rebroadcast content.	VBContent
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned should be examined to determine if an error occurred during the ContentTitleDescriptionUpdate operation.

Method 5:

Syntax:

```
ScheduleEventDecoderDeviceSTBsGet(listOfEncodingTypes, searchText,
sessionID)
```

Inputs:

Name	Description	Type
listOfEncodingTypes	The video encoding types that the target STB(s) needs to decode. This is the value of the ContentEncodingTypes field from the VBScheduledEventContentInfo object returned from the ScheduleEventContentInfoGet method.	List<int>
searchText	Retrieves STB(s) that match text found in the host name, IP address, part number or description. An empty string can be passed to retrieve all STBs available that are capable of decoding the passed encoding types.	string
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBScheduledEventDeviceSTBInfo>
```

The **VBList<VBScheduledEventDeviceSTBInfo>** object returns a list of VBScheduledEventDeviceSTBInfo objects. Each VBScheduledEventDeviceSTBInfo object represents each STB that was matched the encoding types and search text criteria. These objects will become useful when setting up the destination of the scheduled event. The `Exception` field should be examined to determine if any errors occurred during the ScheduleEventDecoderDeviceSTBsGet operation.

Method 6:

Syntax:

```
ScheduleEventAdd(scheduledEventObj, sessionID)
```

Inputs:

Name	Description	Type
scheduledEventObj	The object representing the scheduled event.	VBScheduledEvent
sessionID	The session ID of the user invoking the method.	string

The following fields should be set on the **scheduledEventObj** prior to passing it to the ScheduleEventAdd method:

Name	Description	Type
scheduledEventID	The ID of the scheduled event. This is always set to zero.	int
Description	The description of the scheduled event.	string
Name	The name of the scheduled event.	string
EnableRecurrence	Flag indicating whether or not the event is recurring.	boolean
ShouldDeleteWhenExpired	Flag indicating whether or not to delete the scheduled event when expired.	boolean
EnumScheduledEventTypeID	Always set to the value of the SCHEDULEEVENTTYPE.RebroadcastContentEnum.	int
EnumScheduledEventCreatedBySourceID	Always set to the value of the SCHEDULEDEVENTCREATEDBYSOURCE.Scheduler Enum.	int
EnumTimeZoneID	The ReturnValue field contained in the object returned from DateTimeGetBestGuessEnumTimeZoneID	int
IsTempEvent	Flag indicating whether or not the event is a temporary event. This is usually false.	boolean
MetadataContentID	Set to the ReturnValue in the object returned from ScheduleEventContentSeedCreateFromContenWithMetadata	int
StartDate	Object representing the start date of the scheduled event.	object
StartDateMonth	The start date month (January = 1)	int

Name	Description	Type
StartDateDay	The start date day of the month (1 - 31)	int
StartDateYear	The start date year.	int
StartTimeHour	The start time hour (0 - 23)	int
StartTimeMinute	The start time minute (0 - 59)	int
EndDate	Object representing the end date of the scheduled event.	object
EndDateMonth	The end date month (January = 1)	int
EndDateDay	The end date day of the month. (1 - 31)	int
EndDateYear	The end date year.	int
EndTimeHour	The end time hour (0 - 23)	int
EndTimeMinute	The end time minute (0 - 59)	int
RecurrenceScheduleTimeDurationSeconds	The duration of the scheduled event, in seconds	int
ScheduledEventSource	The object representing the event source. See the following table for details.	VBScheduledEventSource
ScheduledEventDestRecord	The object representing the destination of the recorded content. See the following table for details.	VBScheduledEventDestRecord
ScheduledEventDestinationList	The list representing all destination STBs that the chosen content will play on. This field must be initialized to a new List <VBScheduledEventDestination>( ) prior to use.	List<VBSscheduledEventDestination>

The following fields must be set on the **VBScheduledEventSource** object prior to assigning it to the ScheduledEventSource field of the scheduledEventObj object.

Name	Description	Type
ContentID	The content ID of the original content to rebroadcast.	int
ContentInfo	Set to the object returned from ScheduleEventContentInfoGet.	VBScheduledEventContentInfo

Name	Description	Type
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventSourceID	The ID of the source of the scheduled event. This is always set to zero.	int

The following fields must be set on the **VBScheduledEventDestRecord** object prior to assigning it to the ScheduledEventDestRecord field of the scheduledEventObj object.

Name	Description	Type
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventDestRecordID	The Id of the scheduled event destination record. This is always set to zero.	int
EnumScheduledEventDestRecordTypeID	The destination record type. Since this is not a recording but a playback operation, the SCHEDULEDEVENTDESTRECORDTYPE. None enum is used.	int

The following fields must be set on the **VBScheduledEventDestination** object prior to adding it to the ScheduledEventDestinationList field of the scheduledEventObj object.

Name	Description	Type
EnumScheduledEventDestinationTypeID	This is the destination type ID of the scheduled event. This is always set to the value of the SCHEDULEDEVENTDESTTYPE.STB enum.	int
ScheduledEventID	The ID of the scheduled event destination. This is always set to zero.	int
ScheduledEventDestinationID	The ID of the scheduled event destination. This is always set to zero.	int
DevicesSTBInfo	The is set to the desired VBScheduledEventDevicesSTBInfo object returned from the ScheduleEventDecoderDevicesSTBGet method.	VBScheduledEventDeviceSTBInfo

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ScheduleEventAdd operation by examining the Exception field.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve search criteria from GUI and retrieve the desired content with search
API call (not shown)
VBContent targetContent;

//UTC time zone offset and daylight savings start/end months (could be hardcoded in
config file, etc)
int utcOffsetInMin = -300; //value for eastern United States
int daylightSvgsStartMth = 3; //starts in March
int daylightSvgsEndMth = 11; //ends in November

//1. get time zone info
VBIntData timeZoneData = sll.DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
true, daylightSvgsStartMth, daylightSvgsEndMth, mySession.SessionID);

if (timeZoneData.Exception != null) throw new Exception("Could not get time zone
ID");

//2. retrieve schedule event content info
VBScheduledEventContentInfo eventContentInfo;
eventContentInfo = sll.ScheduleEventContentInfoGet(targetContent.ContentID,
mySession.SessionID);

if (eventContentInfo.Exception != null) throw new Exception("Could not get Content
Info");

//3. generate contentID for rebroadcast
VBIntData rebroadcastContentID =
sll.ScheduleEventContentSeedCreateFromContentWithMetadata(targetContent.ContentID,
mySession.SessionID);

if (rebroadcastContentID.Exception != null) throw new Exception("Could not create
Content ID for rebroadcast");


//4. retrieve STB info
VBList<VBScheduledEventDeviceSTBInfo> allAvailableSTBs =
sll.ScheduleEventDecoderDeviceSTBsGet(eventContentInfo.ContentEncodingTypes,
string.Empty, mySession.SessionID);

if (allAvailableSTBs.Exception != null) throw new Exception("Could not find STBs");

//retrieve title and description content metadata from GUI (not shown here)
string rebroadcastTitle;
string rebroadcastDesc;

//create content object to represent the recording
VBContent rebroadcastContent = new VBContent();
rebroadcastContent.ContentID = targetContent.ContentID;
rebroadcastContent.Title = rebroadcastTitle;
rebroadcastContent.Description = rebroadcastDesc;

//5. update title and description
VBVoidData updateTitleDescResponse =
sll.ContentTitleDescriptionUpdate(rebroadcastContent, mySession.SessionID);

if (updateTitleDescResponse.Exception != null) throw new Exception("Could not
```

```
update title and description of rebroadcast content");

//calculate start and stop times for the rebroadcast (have it start in two minutes)
int targetContentDuration = targetContent.MaxDuration;
DateTime playStartTime = DateTime.Now.AddMinutes(2);
DateTime playStopTime = playStartTime.AddSeconds(targetContentDuration);

//retrieve name and description of the scheduled event from GUI (not shown here)
string eventName;
string eventDesc;

//construct event
VBScheduledEvent newEventToSchedule = new VBScheduledEvent();
newEventToSchedule.ScheduledEventID = 0; //always zero
newEventToSchedule.Description = eventDesc;
newEventToSchedule.Name = eventName;
newEventToSchedule.EnableRecurrence = false; //false if not a recurring event
newEventToSchedule.ShouldDeleteWhenExpired = false;
newEventToSchedule.EnumScheduledEventTypeID =
(int)SCHEDULEEVENTTYPE.RebroadcastContent;
newEventToSchedule.EnumScheduledEventCreatedBySourceID
=(int)SCHEDULEDEVENTCREATEDBYSOURCE.Scheduler;
newEventToSchedule.EnumTimeZoneID = timeZoneData.ReturnValue;
newEventToSchedule.IsTempEvent = false; //always false
newEventToSchedule.MetadataContentID = rebroadcastContentID.ReturnValue;

//set start/end times and duration (should be as long as the content MaxDuration)
newEventToSchedule.StartDate = playStartTime;
newEventToSchedule.StartDateMonth = playStartTime.Month;
newEventToSchedule.StartDateDay = playStartTime.Day;
newEventToSchedule.StartDateYear = playStartTime.Year;
newEventToSchedule.StartTimeHour = playStartTime.Hour;
newEventToSchedule.StartTimeMinute = playStartTime.Minute;
newEventToSchedule.EndDate = playStopTime;
newEventToSchedule.EndDateMonth = playStopTime.Month;
newEventToSchedule.EndDateDay = playStopTime.Day;
newEventToSchedule.EndDateYear = playStopTime.Year;
newEventToSchedule.EndTimeHour = playStopTime.Hour;
newEventToSchedule.EndTimeMinute = playStopTime.Minute;
newEventToSchedule.RecurrenceScheduleTimeDurationSeconds = targetContentDuration;

//supply information about event source
newEventToSchedule.ScheduledEventSource = new VBScheduledEventSource();
newEventToSchedule.ScheduledEventSource.ContentID = targetContent.ContentID; //
original Content's ID       newEventToSchedule.ScheduledEventSource.ContentInfo =
eventContentInfo;         newEventToSchedule.ScheduledEventSource.ScheduledEventID
= 0; //always zero
newEventToSchedule.ScheduledEventSource.ScheduledEventSourceID = 0; //always zero

//supply Destination Record Information
newEventToSchedule.ScheduledEventDestRecord = new VBScheduledEventDestRecord();
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordID = 0; //
always zero
newEventToSchedule.ScheduledEventDestRecord.EnumScheduledEventDestRecordTypeID =
(int)SCHEDULEDEVENTDESTRECORDTYPE.None; //None for play To STB event


//Scheduled Event Destination List
//Let's assume a big list of STBs was returned and we just want to target the 1st
two STBs for //simplicity.  The developer can cherry pick the target STBs as
needed.

//build up the Destination object for the 1st STB
VBScheduledEventDestination firstStbDestination = new
VBScheduledEventDestination();
firstStbDestination.EnumScheduledEventDestinationTypeID =
(int)SCHEDULEDEVENTDESTTYPE.STB;
firstStbDestination.ScheduledEventID = 0; //always zero
firstStbDestination.ScheduledEventDestinationID = 0; //always zero
firstStbDestination.DeviceSTBInfo = allAvailableSTBs.Entities[0];
//build up the Destination object for the 2nd STB
VBScheduledEventDestination secondStbDestination = new
VBScheduledEventDestination();
```

```
secondStbDestination.EnumScheduledEventDestinationTypeID =
(int)SCHEDULEDEVENTDESTTYPE.STB;
secondStbDestination.ScheduledEventID = 0; //always zero
secondStbDestination.ScheduledEventDestinationID = 0; //always zero
secondStbDestination.DeviceSTBInfo = allAvailableSTBs.Entities[1];

//initialize the destination list
newEventToSchedule.ScheduledEventDestinationList = new
List<VBScheduledEventDestination>();

//add the destinations the scheduled event
newEventToSchedule.ScheduledEventDestinationList.Add(firstStbDestination);
newEventToSchedule.ScheduledEventDestinationList.Add(secondStbDestination);

//6. Schedule Rebroadcast Event
VBScheduledEventAddUpdateDeleteReturn scheduleEventResponse;
scheduleEventResponse = sll.ScheduleEventAdd(newEventToSchedule,
newUserSession.SessionID);

if (scheduleEventResponse.Exception != null)
{
throw new Exception("Could not create Rebroadcast Content to STB Scheduled Event");
}

//continue working….

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);

if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
}
```

Live Broadcast Content to STB

The VEMS Mystro API provides the ability to send a live broadcast to a STB. The methods used to create a Live Broadcast scheduled event and send it to a STB will be examined followed by a comprehensive code example.

Method 1:

Syntax:

```
        DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
supportsDaylightSvgsTime, daylightsSavingsTransitionStartMonth,
daylightSavingsTransitionEndMonth, sessionID)
```

Inputs:

Name	Description	Type
utcOffsetInMin	The UTC offset (in minutes) of the target timezone.	int
supportsDaylightSvgsTime	Flag that indicates if the target time zone supports daylight savings time.	boolean
daylightSavingsTransitionStartMonth	The month when daylight savings time begins in the target time zone (January = 1, etc.)	int
daylightSavingsTimeTransitionEndMonth	The month when daylight savings time ends in the target time zone (January = 1, etc.)	int

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned contains the timeZoneID of the target time zone in its ReturnValue field. This value is needed when setting up a scheduled event. The Exception field can also be examined to determine if an error occurred during the DateTimeGetBestGuessEnumTimeZoneID operation.

Method 2:

Syntax:

```
ScheduleEventDeviceSlotsGet(filterText, sessionID, eventSourceTypeList)
```

Inputs:

Name	Description	Type
filterText	Retrieves all device slots that match text found in the host name, IP address, slot name, part number or description. An empty string can be passed to retrieve all slots available within the event source type list criteria.	string
sessionID	The session ID of the user invoking the method.	string
eventSourceTypeList	List of event source types. This is usually the value of the SCHEDULEDEVENTSOURCETYPE .VBrick enum. The other SCHEDULEDEVENTSOURCETHYPE enum types that it could also be are .DME, .RMD, .RMS, or .Content.	List<int>

Output:

```
VBList<VBScheduledEventDeviceSlotInfo>
```

The **VBList<VBScheduledEventDeviceSlotInfo>** object returned is a list of VBScheduledEventDeviceSlotInfo objects that represent all slots that meet the criteria of the filterText and the event source type(s). The `Exception` field should be examined to determine if an error occurred during the ScheduleEventDeviceSlotsGet operation.

Method 3:

Syntax:

```
ScheduleEventDeviceSlotInfoGet(targetSlotID, targetSlotTypeID, sessionID)
```

Inputs:

Name	Description	Type
targetSlotID	The ID of the slot where the live broadcast will originate. Set to the value of the DeviceSlotID field of the chosen VBScheduledEventDevicSlotInfo object returned from ScheduleEventDeviceSlotsGet.	int
targetSlotTypeID	The type of the slot where the live broadcast will originate. Set to the value of the DeviceSlotTypeID field of the chosen VBScheduledEventDeviceSlotInfo object returned from ScheduleEventDeviceSlotGet.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBScheduledEventDeviceSlotInfo object
```

The **VBScheduledEventDeviceSlotInfo** object returned contains information about the target slot that is needed by the VBScheduledEventSource object in the ScheduledEventSource field of the VBScheduledEvent object representing the live broadcast event. The `Exception` field should be examined to determine if an error occurred during the ScheduleEventDeviceSlotInfoGet operation.

Method 4:

Syntax:

```
ScheduleEventContentSeedCreate(sessionID)
```

Inputs:

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned contains the content ID of the live broadcast event in its ReturnValue field. The Exception field should be examined to determine if an error occurred during the ScheduleEventContentSeedCreate operation.

Method 5:

Syntax:

```
ScheduleEventDecoderDeviceSTBsGet(listOfEncodingTypes, searchText,
sessionID)
```

Inputs:

Name	Description	Type
listOfEncoding Types	The video encoding types that the target STB(s) needs to decode. This is the value of the ContentEncodingTypes field from the VBScheduledEventContentInfo object returned from the ScheduleEventContentInfoGet method.	List<int>
searchText	Retrieves STB(s) that match text found in the host name, IP address, part number or description. An empty string can be passed to retrieve all STBs available that are capable of decoding the passed encoding types.	string
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBScheduledEventDeviceSTBInfo>
```

The **VBList<VBScheduledEventDeviceSTBInfo>** object returns a list of VBScheduledEventDeviceSTBInfo objects. Each VBScheduledEventDeviceSTBInfo object represents each STB that was matched the encoding types and search text criteria. These objects will become useful when setting up the destination of the scheduled event. The `Exception` field should be examined to determine if any errors occurred during the ScheduleEventDecoderDeviceSTBsGet operation.

Method 6:

Syntax:

```
ContentTitleDescriptionUpdate(targetContentObj, sessionID)
```

Inputs:

Name	Description	Type
targetContentObj	The object representing the live broadcast content.	VBContent
sessionID	The session ID of the user invoking the method.	string

Method 7:

Syntax:

```
ScheduleEventAdd(scheduledEventObj, sessionID)
```

Inputs:

Name	Description	Type
scheduledEvent Obj	The object representing the scheduled event.	VBScheduledEvent
sessionID	The session ID of the user invoking the method.	string

The following fields should be set on the **scheduledEventObj** prior to passing it to the ScheduleEventAdd method:

Name	Description	Type
scheduledEventID	The ID of the scheduled event. This is always set to zero.	int
Description	The description of the scheduled event.	string
Name	The name of the scheduled event.	string
EnableRecurrence	Flag indicating whether or not the event is recurring.	boolean
ShouldDeleteWhenExpired	Flag indicating whether or not to delete the scheduled event when expired.	boolean
EnumScheduledEventTypeID	Always set to the value of the SCHEDULEEVENTTYPE.RebroadcastContentEnum.	int
EnumScheduledEventCreatedBySourceID	Always set to the value of the SCHEDULEDEVENTCREATEDBYSOURCE.Scheduler Enum.	int
EnumTimeZoneID	The ReturnValue field contained in the object returned from DateTimeGetBestGuessEnumTimeZoneID	int
IsTempEvent	Flag indicating whether or not the event is a temporary event. This is usually false.	boolean
MetadataContentID	Set to the ReturnValue in the object returned from ScheduleEventContentSeedCreateFromContenWithMetadata	int
StartDate	Object representing the start date of the scheduled event.	object
StartDateMonth	The start date month (January = 1)	int
StartDateDay	The start date day of the month (1 - 31)	int
StartDateYear	The start date year.	int
StartTimeHour	The start time hour (0 - 23)	int
StartTimeMinute	The start time minute (0 - 59)	int
EndDate	Object representing the end date of the scheduled event.	object
EndDateMonth	The end date month (January = 1)	int
EndDateDay	The end date day of the month. (1 - 31)	int
EndDateYear	The end date year.	int
EndTimeHour	The end time hour (0 - 23)	int

Name	Description	Type
EndTimeMinute	The end time minute (0 - 59)	int
RecurrenceScheduleTimeDurationSeconds	The duration of the scheduled event, in seconds	int
ScheduledEventSource	The object representing the event source. See the following table for details.	VBScheduledEventSource
ScheduledEventDestRecord	The object representing the destination of the recorded content. See the following table for details.	VBScheduledEventDestRecord
ScheduledEventDestinationList	The list representing all destination STBs that the chosen content will play on. This field must be initialized to a new List <VBScheduledEventDestination>( ) prior to use.	List<VBSscheduledEventDestination>

The following fields must be set on the **VBScheduledEventSource** object prior to assigning it to the ScheduledEventSource field of the scheduledEventObj object.

Name	Description	Type
DeviceSlotID	The value of the DeviceSlotID field in the VBScheduledEventDeviceSlotInfo object returned from ScheduleEventDeviceSlotInfoGet.	int
DeviceSlotInfo	The VBScheduledEventDeviceSlotInfo object returned from ScheduleEventDeviceSlotInfoGet.	VBScheduledEventContentInfo
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventSourceID	The ID of the source of the scheduled event. This is always set to zero.	int

The following fields must be set on the **VBScheduledEventDestRecord** object prior to assigning it to the ScheduledEventDestRecord field of the scheduledEventObj object:

(**Note: This ScheduledEventDestRecord object can be configured to allow the live broadcast to be recorded to a VBStar while it is playing to the target STBs, see the code example for details.)

Name	Description	Type
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventDestRecordID	The Id of the scheduled event destination record. This is always set to zero.	int

Name	Description	Type
EnumScheduledEventDestRecordTypeID	The destination record type. Since this is not a recording but a playback operation, the SCHEDULEDEVENTDESTRECORDTYPE.None enum is used.	int

The following fields must be set on the **VBScheduledEventDestination** object prior to adding it to the ScheduledEventDestinationList field of the scheduledEventObj object.

Name	Description	Type
EnumScheduledEventDestinationTypeID	This is the destination type ID of the scheduled event. This is always set to the value of the SCHEDULEDEVENTDESTTYPE.STB enum.	int
ScheduledEventID	The ID of the scheduled event destination. This is always set to zero.	int
ScheduledEventDestinationID	The ID of the scheduled event destination. This is always set to zero.	int
DevicesSTBInfo	The is set to the desired VBScheduledEventDevicesSTBInfo object returned from the ScheduleEventDecoderDevicesSTBGet method.	VBScheduledEventDeviceSTBInfo

Output:

```
VBVoidData object
```

The **VBVoidData** object returned should be examined to determine if an error occurred during the ScheduleEventAdd operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve search criteria from GUI and retrieve the desired content with search
API call (not shown)
VBContent targetContent;


//UTC time zone offset and daylight savings start/end months (could be hardcoded in
config file, etc)
int utcOffsetInMin = -300; //value for eastern United States
int daylightSvgsStartMth = 3; //starts in March
int daylightSvgsEndMth = 11; //ends in November

//1. get time zone info
VBIntData timeZoneData = sll.DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
true, daylightSvgsStartMth, daylightSvgsEndMth, mySession.SessionID);

if (timeZoneData.Exception != null) throw new Exception("Could not get time zone
ID");

//2. Retrieve the available VBrick slots for the live broadcast
List<int> eventSourceTypes = new List<int>();
eventSourceTypes.Add((int)SCHEDULEDEVENTSOURCETYPE.VBrick); //could also be DME,
RMD,RMS, or Content
VBList<VBScheduledEventDeviceSlotInfo> allAvailableSlots =
sll.ScheduleEventDeviceSlotsGet(string.Empty, mySession.SessionID,
eventSourceTypes);

if (allAvailableSlots.Exception != null) throw new Exception("Could not retrieve
Device Slots");

//3. retrieve info for the desired slot (assume the first slot returned is the
target slot for //simplicity)
VBScheduledEventDeviceSlotInfo targetSlotInfo =
sll.ScheduleEventDeviceSlotInfoGet(allAvailableSlots.Entities[0].DeviceSlotID,
allAvailableSlots.Entities[0].EnumDeviceSlotTypeID, mySession.SessionID);

if (targetSlotInfo.Exception != null) throw new Exception("Could not retrieve
target slot info");

//4. generate content ID for the live broadcast
VBIntData liveBroadcastContentID =
sll.ScheduleEventContentSeedCreate(mySession.SessionID);

if (liveBroadcastContentID.Exception != null) throw new Exception("Could not create
Content ID");

//create list of encoding types based on encoding type that the target slot
supports
List<int> targetEncodingTypes = new List<int>();
targetEncodingTypes.Add(targetSlotInfo.EnumEncodingTypeID);
//5. Retrieve all STB info
VBList<VBScheduledEventDeviceSTBInfo> allAvailableSTBs =
sll.ScheduleEventDecoderDeviceSTBsGet(targetEncodingTypes, string.Empty,
mySession.SessionID);

//retrieve live broadcast title and description from GUI (not shown here)
string liveBroadcastTitle;
string liveBroadcastDescription;

//6. update title and description
VBContent liveBroadcastContent = new VBContent();
liveBroadcastContent.ContentID = liveBroadcastContentID.ReturnValue;
liveBroadcastContent.Title = liveBroadcastTitle;
liveBroadcastContent.Description = liveBroadcastDescription;
VBVoidData updateTitleDescResponse =
sll.ContentTitleDescriptionUpdate(liveBroadcastContent, mySession.SessionID);
```

```
if (updateTitleDescResponse.Exception != null) throw new Exception("Could not
update title and description");

//retrieve the length of broadcast in minutes from GUI (not shown here)
int broadcastInMinutes;
DateTime broadcastStartTime = DateTime.Now;
DateTime broadcastEndTime = DateTime.Now.AddMinutes(broadcastInMinutes);
int durationOfBroadcastInSeconds = broadcastInMinutes * 60;

//retrieve name and description of the scheduled event from GUI (not shown here)
string name;
string description;

//construct event
VBScheduledEvent newEventToSchedule = new VBScheduledEvent();
newEventToSchedule.ScheduledEventID = 0; //always zero
newEventToSchedule.Description = description;
newEventToSchedule.Name = name;
newEventToSchedule.EnableRecurrence = false; //false if not a recurring event.
newEventToSchedule.ShouldDeleteWhenExpired = false;
newEventToSchedule.EnumScheduledEventTypeID =
(int)SCHEDULEEVENTTYPE.LiveBroadcast;
newEventToSchedule.EnumScheduledEventCreatedBySourceID =
(int)SCHEDULEDEVENTCREATEDBYSOURCE.Scheduler;
newEventToSchedule.EnumTimeZoneID = timeZoneData.ReturnValue;
newEventToSchedule.IsTempEvent = false; //always false
newEventToSchedule.MetadataContentID = liveBroadcastContentID.ReturnValue;

//set start/end times and duration
newEventToSchedule.StartDate = broadcastStartTime;
newEventToSchedule.StartDateMonth = broadcastStartTime.Month;
newEventToSchedule.StartDateDay = broadcastStartTime.Day;
newEventToSchedule.StartDateYear = broadcastStartTime.Year;
newEventToSchedule.StartTimeHour = broadcastStartTime.Hour;
newEventToSchedule.StartTimeMinute = broadcastStartTime.Minute;
newEventToSchedule.EndDate = broadcastEndTime;
newEventToSchedule.EndDateMonth = broadcastEndTime.Month;
newEventToSchedule.EndDateDay = broadcastEndTime.Day;
newEventToSchedule.EndDateYear = broadcastEndTime.Year;
newEventToSchedule.EndTimeHour = broadcastEndTime.Hour;
newEventToSchedule.EndTimeMinute = broadcastEndTime.Minute;
newEventToSchedule.RecurrenceScheduleTimeDurationSeconds =
durationOfBroadcastInSeconds;

//supply information about event source
newEventToSchedule.ScheduledEventSource = new VBScheduledEventSource();
newEventToSchedule.ScheduledEventSource.DeviceSlotID =
targetSlotInfo.DeviceSlotID;
newEventToSchedule.ScheduledEventSource.DeviceSlotInfo = targetSlotInfo;
newEventToSchedule.ScheduledEventSource.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventSource.ScheduledEventSourceID = 0; //always zero

//supply Destination Record Information
//NOTE: This is how the Event Destination Record is configured when just playing
the live broadcast //to a STB.  The highlighted code block shown below shows how the
destination record should be //configured if the live broadcast is being sent to a
STB AND being recorded by a VBStar.
newEventToSchedule.ScheduledEventDestRecord = new VBScheduledEventDestRecord();
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordID = 0; //
always zero
newEventToSchedule.ScheduledEventDestRecord.EnumScheduledEventDestRecordTypeID =
(int)SCHEDULEDEVENTDESTRECORDTYPE.None; //None for play To STB scheduled event


//Note: set up the destination record as seen in this code block to record the live
broadcast to a //VBStar (It will still be sent to the target STBs).  This sample
code block assumes that the //targetSlotInfo is from a VBStar.
//<RecordToVBStar>
newEventToSchedule.ScheduledEventDestRecord = new VBScheduledEventDestRecord();
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordID = 0; //
always zero
newEventToSchedule.ScheduledEventDestRecord.DeviceVBrickSlotID =
```

```
targetSlotInfo.DeviceSlotID;
newEventToSchedule.ScheduledEventDestRecord.EnumScheduledEventDestRecordTypeID =
(int)SCHEDULEDEVENTDESTRECORDTYPE.VBrick; //must be VBrick to record to VBStar
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick = new
VBScheduledEventDestRecordVBrick();
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick.DeviceV
BrickSlotID = targetSlotInfo.DeviceSlotID;
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick.Schedul
edEventDestRecordID = 0; //always zero
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick.ShouldF
tpAfterRecord = true; //always true
//</RecordToVBStar>

//Scheduled Event Destination List (let's just add the 1st two STBs for simplicity)
//The developer can cherry pick STBs as needed.  Each STB requires its own //
VBScheduledEventDestination object.

//build up the Destination object for the 1st STB
VBScheduledEventDestination firstStbDestination = new
VBScheduledEventDestination();
firstStbDestination.EnumScheduledEventDestinationTypeID =
(int)SCHEDULEDEVENTDESTTYPE.STB;
firstStbDestination.ScheduledEventID = 0; //always zero
firstStbDestination.ScheduledEventDestinationID = 0; //always zero
firstStbDestination.DeviceSTBInfo = allAvailableSTBs.Entities[0];

//build up the Destination object for the 2nd STB
VBScheduledEventDestination secondStbDestination = new
VBScheduledEventDestination();
secondStbDestination.EnumScheduledEventDestinationTypeID =
(int)SCHEDULEDEVENTDESTTYPE.STB;
secondStbDestination.ScheduledEventID = 0;
secondStbDestination.ScheduledEventDestinationID = 0;
secondStbDestination.DeviceSTBInfo = allAvailableSTBs.Entities[1];

//initialize the destination list
newEventToSchedule.ScheduledEventDestinationList = new
List<VBScheduledEventDestination>();
//add the destinations the scheduled event
newEventToSchedule.ScheduledEventDestinationList.Add(firstStbDestination);
newEventToSchedule.ScheduledEventDestinationList.Add(secondStbDestination);

//7. Schedule Event
VBScheduledEventAddUpdateDeleteReturn scheduleEventResponse;
scheduleEventResponse = sll.ScheduleEventAdd(newEventToSchedule,
mySession.SessionID);

if (scheduleEventResponse.Exception != null) throw new Exception("Could not
schedule the live broadcast");

//continue working….

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("Could not log out");
```

Tune STB to Existing Stream

The VEMS Mystro API provides the ability to tune an existing Live stream to a STB. A scheduled event must be created in order to tune a STB to a Live stream. The methods involved with creating a Tune STB to Existing Stream scheduled event will be explored followed by a comprehensive code example.

Method 1:

Syntax:

```
        DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
supportsDaylightSvgsTime, daylightsSavingsTransitionStartMonth,
daylightSavingsTransitionEndMonth, sessionID)
```

Inputs:

Name	Description	Type
utcOffsetInMin	The UTC offset (in minutes) of the target timezone.	int
supportsDaylightSvgsTime	Flag that indicates if the target time zone supports daylight savings time.	boolean
daylightSavingsTransitionStartMonth	The month when daylight savings time begins in the target time zone (January = 1, etc.)	int
daylightSavingsTimeTransitionEndMonth	The month when daylight savings time ends in the target time zone (January = 1, etc.)	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned contains the timeZoneID of the target time zone in its `ReturnValue` field. This value is needed when setting up a scheduled event. The `Exception` field can also be examined to determine if an error occurred during the DateTimeGetBestGuessEnumTimeZoneID operation.

Method 2:

Syntax:

```
ScheduleEventContentInfoGet(targetContentID, sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the content to rebroadcast.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBScheduledEventContentInfo object
```

The **VBScheduledEventContentInfo** object returned contains information about the content to rebroadcast that the scheduling module needs. The `Exception` field should be examined to determine if an error occurred during the ScheduleEventContentInfoGet operation.

Method 3:

Syntax:

```
        ScheduleEventContentSeedCreateFromContentWithMetadata(targetContentID,
sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the content to record.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
        VBIntData object
```

The **VBIntData** object returned contains the newly generated content ID for the rebroadcast content event in its `ReturnValue` field. The `Exception` field should be examined to determine if an error occurred during the ScheduleEventContentSeedCreateFromContentWithMetadata operation.

Method 4:

Syntax:

```
        ScheduleEventDecoderDeviceSTBsGet(listOfEncodingTypes, searchText,
sessionID)
```

Inputs:

Name	Description	Type
listOfEncoding Types	The video encoding types that the target STB(s) needs to decode. This is the value of the ContentEncodingTypes field from the VBScheduledEventContentInfo object returned from the ScheduleEventContentInfoGet method.	List<int>
searchText	Retrieves STB(s) that match text found in the host name, IP address, part number or description. An empty string can be passed to retrieve all STBs available that are capable of decoding the passed encoding types.	string
sessionID	The session ID of the user invoking the method.	string

Output:

```
        VBList<VBScheduledEventDeviceSTBInfo>
```

The **VBList<VBScheduledEventDeviceSTBInfo>** object returns a list of VBScheduledEventDeviceSTBInfo objects. Each VBScheduledEventDeviceSTBInfo object represents each STB that was matched the encoding types and search text criteria. These

objects will become useful when setting up the destination of the scheduled event. The `Exception` field should be examined to determine if any errors occurred during the ScheduleEventDecoderDeviceSTBsGet operation.

Method 5:

Syntax:

```
ScheduleEventAdd(scheduledEventObj, sessionID)
```

Inputs:

Name	Description	Type
scheduledEventObj	The object representing the scheduled event.	VBScheduledEvent
sessionID	The session ID of the user invoking the method.	string

The following fields should be set on the **scheduledEventObj** prior to passing it to the ScheduleEventAdd method:

Name	Description	Type
scheduledEventID	The ID of the scheduled event. This is always set to zero.	int
Description	The description of the scheduled event.	string
Name	The name of the scheduled event.	string
EnableRecurrence	Flag indicating whether or not the event is recurring.	boolean
ShouldDeleteWhenExpired	Flag indicating whether or not to delete the scheduled event when expired.	boolean
EnumScheduledEventTypeID	Always set to the value of the SCHEDULEEVENTTYPE.RebroadcastContentEnum.	int
EnumScheduledEventCreatedBySourceID	Always set to the value of the SCHEDULEDEVENTCREATEDBYSOURCE.Scheduler Enum.	int
EnumTimeZoneID	The ReturnValue field contained in the object returned from DateTimeGetBestGuessEnumTimeZoneID	int
IsTempEvent	Flag indicating whether or not the event is a temporary event. This is usually false.	boolean
MetadataContentID	Set to the ReturnValue in the object returned from ScheduleEventContentSeedCreateFromContenWithMetadata	int

Name	Description	Type
StartDate	Object representing the start date of the scheduled event.	object
StartDateMonth	The start date month (January = 1)	int
StartDateDay	The start date day of the month (1 - 31)	int
StartDateYear	The start date year.	int
StartTimeHour	The start time hour (0 - 23)	int
StartTimeMinute	The start time minute (0 - 59)	int
EndDate	Object representing the end date of the scheduled event.	object
EndDateMonth	The end date month (January = 1)	int
EndDateDay	The end date day of the month. (1 - 31)	int
EndDateYear	The end date year.	int
EndTimeHour	The end time hour (0 - 23)	int
EndTimeMinute	The end time minute (0 - 59)	int
RecurrenceScheduleTimeDurationSeconds	The duration of the scheduled event, in seconds	int
ScheduledEventSource	The object representing the event source. See the following table for details.	VBScheduledEventSource
ScheduledEventDestRecord	The object representing the destination of the recorded content. See the following table for details.	VBScheduledEventDestRecord
ScheduledEventDestinationList	The list representing all destination STBs that the chosen content will play on. This field must be initialized to a new List <VBScheduledEventDestination>( ) prior to use.	List<VBSscheduledEventDestination>

The following fields must be set on the **VBScheduledEventSource** object prior to assigning it to the ScheduledEventSource field of the scheduledEventObj object.

Name	Description	Type
ContentID	The content ID of the original content to rebroadcast.	int
ContentInfo	Set to the object returned from ScheduleEventContentInfoGet.	VBScheduledEventContentInfo
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventSourceID	The ID of the source of the scheduled event. This is always set to zero.	int

The following fields must be set on the **VBScheduledEventDestRecord** object prior to assigning it to the ScheduledEventDestRecord field of the scheduledEventObj object.

Name	Description	Type
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventDestRecordID	The Id of the scheduled event destination record. This is always set to zero.	int
EnumScheduledEventDestRecordTypeID	The destination record type. Since this is not a recording but a playback operation, the SCHEDULEDEVENTDESTRECORDTYPE.None enum is used.	int

The following fields must be set on the **VBScheduledEventDestination** object prior to adding it to the ScheduledEventDestinationList field of the scheduledEventObj object.

Name	Description	Type
EnumScheduledEventDestinationTypeID	This is the destination type ID of the scheduled event. This is always set to the value of the SCHEDULEDEVENTDESTTYPE.STB enum.	int
ScheduledEventID	The ID of the scheduled event destination. This is always set to zero.	int
ScheduledEventDestinationID	The ID of the scheduled event destination. This is always set to zero.	int
DevicesSTBInfo	The is set to the desired VBScheduledEventDevicesSTBInfo object returned from the ScheduleEventDecoderDevicesSTBGet method.	VBScheduledEventDeviceSTBInfo

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ScheduleEventAdd operation by examining the Exception field.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");
```

```
//retrieve search criteria from GUI and retrieve the desired content with search
API call (not shown)
VBContent targetContent;


//UTC time zone offset and daylight savings start/end months (could be hardcoded in
config file, etc)
int utcOffsetInMin = -300; //value for eastern United States
int daylightSvgsStartMth = 3; //starts in March
int daylightSvgsEndMth = 11; //ends in November

//1. get time zone info
VBIntData timeZoneData = sll.DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
true, daylightSvgsStartMth, daylightSvgsEndMth, mySession.SessionID);

if (timeZoneData.Exception != null) throw new Exception("Could not get time zone
ID");

//retrieve search criteria from GUI, perform search and retrieve target content
(not shown here)
VBContent targetContent;

//2. Retrieve content info
VBScheduledEventContentInfo liveStreamInfo;
liveStreamInfo = sll.ScheduleEventContentInfoGet(targetContent.ContentID,
mySession.SessionID);

if (liveStreamInfo.Exception != null) throw new Exception("Could not retrieve
content info");

//3. Create Content ID for the 'Tune STB to Existing Stream' event
VBIntData tuneSTBToStreamEventContentID =
sll.ScheduleEventContentSeedCreateFromContentWithMetadata(targetContent.ContentID,
mySession.SessionID);

if (tuneSTBToStreamEventContentID.Exception != null)
{
throw new Exception("Coult not generate ContentID for event");
}

//4. retrieve STB info
VBList<VBScheduledEventDeviceSTBInfo> allAvailableSTBs =
sll.ScheduleEventDecoderDeviceSTBsGet(liveStreamInfo.ContentEncodingTypes,
string.Empty, newUserSession.SessionID);

if (allAvailableSTBs.Exception != null) throw new Exception("Could not retrieve STB
info");

//retrieve length of time for STB to tune in for from the GUI (not shown here)
int numOfMinutesToTuneInFor;
DateTime tuneSTBStart = DateTime.Now;
DateTime tuneSTBEnd = DateTime.Now.AddMinutes(numOfMinutesToTuneInFor);
int durationToTuneInFor = numOfMinutesToTuneInFor * 60; //convert minutes to
seconds

//retrieve the name and description of the Tune STB to Existing stream event from
GUI (not shown)
string name;
string description;

//construct event
VBScheduledEvent newEventToSchedule = new VBScheduledEvent();
newEventToSchedule.ScheduledEventID = 0; //always zero
newEventToSchedule.Description = description;
newEventToSchedule.Name = name;
newEventToSchedule.EnableRecurrence = false; //false if not a recurring event.
newEventToSchedule.ShouldDeleteWhenExpired = false;
newEventToSchedule.EnumScheduledEventTypeID = (int)SCHEDULEEVENTTYPE.TuneSTB;
newEventToSchedule.EnumScheduledEventCreatedBySourceID =
(int)SCHEDULEDEVENTCREATEDBYSOURCE.Scheduler;
newEventToSchedule.EnumTimeZoneID = timeZoneData.ReturnValue;
newEventToSchedule.IsTempEvent = false; //always false
newEventToSchedule.MetadataContentID = tuneSTBToStreamEventContentID.ReturnValue;
//new content ID
```

```
//set start/end times and duration
newEventToSchedule.StartDate = tuneSTBStart;
newEventToSchedule.StartDateMonth = tuneSTBStart.Month;
newEventToSchedule.StartDateDay = tuneSTBStart.Day;
newEventToSchedule.StartDateYear = tuneSTBStart.Year;
newEventToSchedule.StartTimeHour = tuneSTBStart.Hour;
newEventToSchedule.StartTimeMinute = tuneSTBStart.Minute;
newEventToSchedule.EndDate = tuneSTBEnd;
newEventToSchedule.EndDateMonth = tuneSTBEnd.Month;
newEventToSchedule.EndDateDay = tuneSTBEnd.Day;
newEventToSchedule.EndDateYear = tuneSTBEnd.Year;
newEventToSchedule.EndTimeHour = tuneSTBEnd.Hour;
newEventToSchedule.EndTimeMinute = tuneSTBEnd.Minute;
newEventToSchedule.RecurrenceScheduleTimeDurationSeconds = durationToTuneInFor;

//supply information about event source
newEventToSchedule.ScheduledEventSource = new VBScheduledEventSource();
newEventToSchedule.ScheduledEventSource.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventSource.ScheduledEventSourceID = 0; //always zero
newEventToSchedule.ScheduledEventSource.ContentID = targetContent.ContentID; //
original contentID
newEventToSchedule.ScheduledEventSource.ContentInfo = liveStreamInfo;

//supply information about event Destination
newEventToSchedule.ScheduledEventDestRecord = new VBScheduledEventDestRecord();
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordID = 0; //
always zero
newEventToSchedule.ScheduledEventDestRecord.EnumScheduledEventDestRecordTypeID =
(int)SCHEDULEDEVENTDESTRECORDTYPE.None; //None for play To STB scheduled event

//Scheduled Event Destination List (let's just add the 1st two STBs for simplicity)
//build up the Destination object for the 1st STB
VBScheduledEventDestination firstStbDestination = new
VBScheduledEventDestination();
firstStbDestination.EnumScheduledEventDestinationTypeID =
(int)SCHEDULEDEVENTDESTTYPE.STB;
firstStbDestination.ScheduledEventID = 0; //always zero
firstStbDestination.ScheduledEventDestinationID = 0; //always zero
firstStbDestination.DeviceSTBInfo = allAvailableSTBs.Entities[0];
//build up the Destination object for the 2nd STB
VBScheduledEventDestination secondStbDestination = new
VBScheduledEventDestination();
secondStbDestination.EnumScheduledEventDestinationTypeID =
(int)SCHEDULEDEVENTDESTTYPE.STB;
secondStbDestination.ScheduledEventID = 0;
secondStbDestination.ScheduledEventDestinationID = 0;
secondStbDestination.DeviceSTBInfo = allAvailableSTBs.Entities[1];
//initialize the destination list
newEventToSchedule.ScheduledEventDestinationList = new
List<VBScheduledEventDestination>();
//add the destinations the scheduled event
newEventToSchedule.ScheduledEventDestinationList.Add(firstStbDestination);
newEventToSchedule.ScheduledEventDestinationList.Add(secondStbDestination);
//6. Schedule Event
VBScheduledEventAddUpdateDeleteReturn scheduleEventResponse;
scheduleEventResponse = sll.ScheduleEventAdd(newEventToSchedule,
newUserSession.SessionID);

if (scheduleEventResponse.Exception != null)
{
throw new Exception("Could not add Tune STB To Existing Stream event");
}


//continue working….


//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("Logout failed");
```

Chapter 11

Record Content

Topics in this document

Recording Content 101
Determine if Content is Recordable 101
Start Recording Content 102
Edit Recording Metadata 103
Stop Recording Content. 106
Get Record Request Status. 106
Record to VBStar 108

Recording Content

The VEMS Mystro API provides the ability to record live content being viewed by the user. Only *live* content may be recorded. The recorded subset of the original content is treated just like any other content; it is searchable and can have metadata associated with it. Note that not all live content can be recorded. Prior to beginning a recording, a method is used to first determine if a unit of content is recordable. After a unit of content has been verified as recordable, the method used to start a recording is explored, followed by the methods needed to assign metadata to the recorded content, retrieve the recording status and finally, stop the recording.

Determine if Content is Recordable

The VEMS Mystro API provides the ability to determine if a unit of content is recordable. For technical reasons, not all content can be recorded. It is good practice to determine if a particular unit of content is recordable prior to attempting the record operation. There is one method responsible for determining if a unit of content is recordable.

Syntax:

```
CanRecord(targetContentObj, sessionID)
```

Inputs:

Name	Description	Type
targetContentObj	The object representing the content that the user wants to record.	VBContent
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBBoolData object
```

The **VBBoolData** object returned allows client code to determine if the target content is recordable by examining its `ReturnValue` field. If the ReturnValue field is true, the content is recordable otherwise it is not recordable. The `Exception` field should also be examined to determine if an error occurred during the CanRecord operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content to test by doing a search based on input from GUI (not shown)
string targetContentTitle; //from GUI
VBContent targetContent; //populate with search result

//determine if the content is recordable
VBBoolData isTargetContentRecordable;
isTargetContentRecordable = sll.CanRecord(targetContent, mySession.SessionID);

if (isTargetContentRecordable.Exception != null) throw new Exception("CanRecord
failed");

if (isTargetContentRecordable.ReturnValue)
{
//content is recordable, take action (start recording)
}
else
{
//content is not recordable
throw new Exception("Content is not recordable");
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
```

Start Recording Content

The VEMS Mystro API provides the ability to start recording live content that is playing. There is one method responsible for recording content, **ContentRecordStart**. ContentRecordStart has 3 overloads, however only one will be examined below.

Syntax:

```
ContentRecordStart(targetContentID, contentObj, sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the content to record.	int
contentObj	The object representing a unit of content (pass in new VBContent object).	VBContent
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBRecordInfo object
```

The **VBRecordInfo** object returned allows client code to update the metadata associated with the recording by using the VBRecordRequest object in the `Record` field. The `Exception` field should be examined to determine if an error occurred during the ContentRecordStart operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content to test by doing a search based on input from GUI (not shown)
string targetContentTitle; //from GUI
VBContent targetContent; //populate with search result

//determine if the content is recordable
VBBoolData isTargetContentRecordable;
isTargetContentRecordable = sll.CanRecord(targetContent, mySession.SessionID);

if (isTargetContentRecordable.Exception != null) throw new Exception("CanRecord
failed");

VBRecordInfo contentRecordInfo;
if (isTargetContentRecordable.ReturnValue)
{
//content is recordable
contentRecordInfo = sll.ContentRecordStart(targetContent.ContentID, new
VBContent(),          mySession.SessionID);
if (contentRecordInfo.Exception != null) throw new Exception("ContentRecordStart
failed");

}
else
{
throw new Exception("Content is not recordable");
}

//let content continue recording
//eventually stop recording and log out (not shown here)
```

Edit Recording Metadata

The VEMS Mystro API provides the ability to edit the metadata associated with a recording. There is one method responsible for updating recorded content metadata; it's called **ContentRecordUpdateMetadata**.

There is another supporting method which retrieves all categories that the invoking user has Add access to. This will come in handy when it comes time to select the categories to add the recorded content to. This method is called **ContentCategoriesAvailableForEdit.** An example incorporating both methods will be shown after examining each.

Method 1:

Syntax:

```
ContentRecordUpdateMetadata(contentObj, requestRecordObj, sessionID)
```

Inputs:

Name	Description	Type
contentObj	The object representing the recorded content.	VBContent
requestRecordedObj	An object representing the record request (this can be found in the VBRecordInfo object returned from ContentRecordStart).	VBRequestRecord
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned allows client code to determine if an error occurred during the ContentRecordUpdateMetadata operation by examining the `Exception` field.

Method 2:

Syntax:

```
ContentCategoriesGetAvailableForEdit(targetContentID, sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the original content being recorded.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBList<VBCategory>
```

The **VBList<VBCategory>** object returned is a list of VBCategory objects representing all categories that the invoking user has **Add** access to. The recorded content may be added to any of these categories. The `Exception` field should be examined to determine if an error occurred during the ContentCategoriesGetAvailableForEdit operation.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content by doing a search based on input from GUI (not shown)
string targetContentTitle; //from GUI
VBContent targetContent; //populate with search result
```

```
//determine if the content is recordable
VBBoolData isTargetContentRecordable;
isTargetContentRecordable = sll.CanRecord(targetContent, mySession.SessionID);

if (isTargetContentRecordable.Exception != null) throw new Exception("CanRecord
failed");
if (!isTargetContentRecordable.ReturnValue) throw new Exception("Content is not
recordable");

//start recording
VBRecordInfo contentRecordInfo;
contentRecordInfo = sll.ContentRecordStart(targetContent.ContentID, new
VBContent(),         mySession.SessionID);

if (contentRecordInfo.Exception != null) throw new Exception("Content Record
failed");

//now content metadata can be updated

//get the title, description and name of category to add recorded content to, from
GUI (not shown)
string categoryName;
string title;
string description;

//retrieve all categories available
VBList<VBCategory> allCat =
sll.ContentCategoriesAvailableForEdit(targetContent.ContentID,
mySession.SessionID);
//make sure this category is on the list
VBCategory targetCategory = allCat.Entities.Find(myCategory => myCategory.Name ==
categoryName);
if (targetCategory == null) throw new Exception("target category does not exist");

//create new category object with just the target category's ID and Name
VBCategory catForMetadataUpdate = new VBCategory();
catForMetadataUpdate.CategoryID = targetCategory.CategoryID;
catForMetadataUpdate.Name = targetCategory.Name;

//create new content object
VBContent recordedContent = new VBContent();
recordedContent.Title = title;
recordedContent.Description = description;
recordedContent.Categories = new List<VBCategory>();
recordedContent.Categories.Add(catForMetadataUpdate);

//make the call to update the metadata
VBVoidData updateRecordingMetadata;
updateRecordingMetadata =
sll.ContentRecordUpdateMetadata(contentRecordInfo.Record, recordedContent,
mySession.SessionID);

if (updateRecordingMetadata.Exception != null) throw new Exception("Recording
metadata update failed");

//continue working and logout (not shown)
```

Stop Recording Content

The VEMS Mystro API provides the ability to stop recording content. There is one method responsible for stopping a recording, that method is called **ContentRecordStop**. The ContentRecordStop method is overloaded; only one of the overloads is discussed below.

Syntax:

```
ContentRecordStop(targetRecordRequestObj, sessionID)
```

Inputs:

Name	Description	Type
targetRecordRequestObj	The object representing the record request. (This is from the VBRecordInfo object returned from ContentRecordStart)	(VBRequestRecord)
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBRequestRecord object
```

The **VBRequestRecord** object returned can be used to determine if an error occurred during the ContentRecordStop operation by examining its `Exception` field. The RequestID field can also be used when calling another method to determine the status of the record request.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//retrieve content by doing a search based on input from GUI (not shown)
string targetContentTitle; //from GUI
VBContent targetContent; //populate with search result

//determine if the content is recordable
VBBoolData isTargetContentRecordable;
isTargetContentRecordable = sll.CanRecord(targetContent, mySession.SessionID);

if (isTargetContentRecordable.Exception != null) throw new Exception("CanRecord
failed");
if (!isTargetContentRecordable.ReturnValue) throw new Exception("Content is not
recordable");

//start recording
VBRecordInfo contentRecordInfo;
contentRecordInfo = sll.ContentRecordStart(targetContent.ContentID, new
VBContent(),          mySession.SessionID);

if (contentRecordInfo.Exception != null) throw new Exception("Content Record
failed");

//add record content metadata, wait some time until all desired content has been
recorded

//stop recording
VBRequestRecord stopRecordingResponse =
```

```
sll.ContentRecordStop(contentRecordInfo.Record, mySession.SessionID);

if (stopRecordingResponse.Exception != null) throw new Exception("Stop Recording
Content failed");
//continue on and eventually logout (not shown here)
```

Get Record Request Status

The VEMS Mystro API provides the ability to monitor the status of a record request and ensure that it terminates normally. The method used to check record requests is called **ContentRecordStatusGet**.

Syntax:

```
ContentRecordStatusGet(recordRequestID, sessionID)
```

Inputs:

Name	Description	Type
recordRequestID	The ID of the target record request (found within the RequestID field of the VBRecordInfo object returned from ContentRecordStart)	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBRequestRecord object
```

The **VBRequestRecord** object returned can be used to determine if an error occurred during the ContentRecordStatusGet operation by examining its `Exception` field.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed.  See the
ExceptionLog table for details");

//get input from GUI to search for content (not shown)
//perform search for target content based on input from GUI (not shown)
VBContent targetContent; //populated from search results

//determine if the content is recordable
VBBoolData isTargetContentRecordable;
isTargetContentRecordable = sll.CanRecord(targetContent, mySession.SessionID);

if (isTargetContentRecordable.Exception != null) throw new Exception("CanRecord
failed");
if (!isTargetContentRecordable.ReturnValue) throw new Exception("Content is not
recordable");

//start recording
VBRecordInfo contentRecordInfo;
contentRecordInfo = sll.ContentRecordStart(targetContent.ContentID, new
VBContent(),         mySession.SessionID);

if (contentRecordInfo.Exception != null) throw new Exception("Content Record
failed");

//add record content metadata, wait some time until all desired content has been
recorded
```

```
//stop recording
VBRequestRecord stopRecordingResponse =
sll.ContentRecordStop(contentRecordInfo.Record, mySession.SessionID);

if (stopRecordingResponse.Exception != null) throw new Exception("Stop Recording
Content failed");

//get status of record request and verify that it finished without error
VBRequestRecord finishRecordingResponse =
sll.ContentRecordStatusGet(contentRecordInfo.Record.RequestID,
mySession.SessionID);
if (finishRecordingResponse.Exception != null) throw new Exception("Record Request
Status Retrieval failed");


//verify the status
if (finishRecordingResponse.StatusDescription == "Record completed")
{
//the recording successfully finished
}

//can also check the PercentDone field

if (finisheRecordingResponse.PercentDone == 100)
{
//the recording successfully finished
}

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
```

Record to VBStar

The VEMS Mystro API provides the ability to record a Live stream to a VBStar. The live stream to record must originate from a VBStar. A scheduled event must be created in order to record to a VBStar. The methods involved with creating a Record scheduled event will be discussed below, followed by a comprehensive code example.

Method 1:

Syntax:

```
        DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
supportsDaylightSvgsTime, daylightsSavingsTransitionStartMonth,
daylightSavingsTransitionEndMonth, sessionID)
```

Inputs:

Name	Description	Type
utcOffsetInMin	The UTC offset (in minutes) of the target timezone.	int
supportsDaylightSvgsTime	Flag that indicates if the target time zone supports daylight savings time.	boolean
daylightSavingsTransitionStartMonth	The month when daylight savings time begins in the target time zone (January = 1, etc.)	int
daylightSavingsTimeTransitionEndMonth	The month when daylight savings time ends in the target time zone (January = 1, etc.)	int

Name	Description	Type
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned contains the timeZoneID of the target time zone in its `ReturnValue` field. This value is needed when setting up a scheduled event. The `Exception` field can also be examined to determine if an error occurred during the DateTimeGetBestGuessEnumTimeZoneID operation.

Method 2:

Syntax:

```
ScheduleEventContentInfoGet(targetContentID, sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the live content to record.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBScheduledEventContentInfo object
```

The **VBScheduledEventContentInfo** object returned contains information about the content to rebroadcast that the scheduling module needs. The `Exception` field should be examined to determine if an error occurred during the ScheduleEventContentInfoGet operation.

Method 3:

Syntax:

```
ScheduleEventContentSeedCreateFromContentWithMetadata(targetContentID,
sessionID)
```

Inputs:

Name	Description	Type
targetContentID	The content ID of the content to record.	int
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBIntData object
```

The **VBIntData** object returned contains the newly generated content ID for the Record event in its **`ReturnValue`** field. The **`Exception`** field should be examined to determine if an error occurred during the ScheduleEventContentSeedCreateFromContentWithMetadata operation.

Method 4:

Syntax:

```
ContentTitleDescriptionUpdate(targetContentObj, sessionID)
```

Inputs:

Name	Description	Type
targetContentObj	The object representing the recorded content.	VBContent
sessionID	The session ID of the user invoking the method.	string

Output:

```
VBVoidData object
```

The **VBVoidData** object returned should be examined to determine if an error occurred during the ContentTitleDescriptionUpdate operation.

Method 5:

Syntax:

```
ScheduleEventAdd(scheduledEventObj, sessionID)
```

Inputs:

Name	Description	Type
scheduledEventObj	The object representing the scheduled recording event.	VBScheduledEvent
sessionID	The session ID of the user invoking the method.	string

The following fields should be set on the **scheduledEventObj** prior to passing it to the ScheduleEventAdd method:

Name	Description	Type
scheduledEventID	The ID of the scheduled event. This is always set to zero.	int
Description	The description of the scheduled event.	string
Name	The name of the scheduled event.	string
EnableRecurrence	Flag indicating whether or not the event is recurring.	boolean

Name	Description	Type
ShouldDeleteWhen Expired	Flag indicating whether or not to delete the scheduled event when expired.	boolean
EnumScheduledEve ntTypeID	Always set to the value of the SCHEDULEEVENTTYPE.Rebroadc astContentEnum.	int
EnumScheduledEve ntCreatedBySourceI D	Always set to the value of the SCHEDULEDEVENTCREATEDBY SOURCE.Scheduler Enum.	int
EnumTimeZoneID	The ReturnValue field contained in the object returned from DateTimeGetBestGuessEnumTimeZo neID	int
IsTempEvent	Flag indicating whether or not the event is a temporary event. This is usually false.	boolean
MetadataContentID	Set to the ReturnValue in the object returned from ScheduleEventContentSeedCreateFro mContenWithMetadata	int
StartDate	Object representing the start date of the scheduled event.	object
StartDateMonth	The start date month (January = 1)	int
StartDateDay	The start date day of the month (1 - 31)	int
StartDateYear	The start date year.	int
StartTimeHour	The start time hour (0 - 23)	int
StartTimeMinute	The start time minute (0 - 59)	int
EndDate	Object representing the end date of the scheduled event.	object
EndDateMonth	The end date month (January = 1)	int
EndDateDay	The end date day of the month. (1 - 31)	int
EndDateYear	The end date year.	int
EndTimeHour	The end time hour (0 - 23)	int
EndTimeMinute	The end time minute (0 - 59)	int
RecurrenceSchedule TimeDurationSecon ds	The duration of the scheduled event, in seconds	int

Name	Description	Type
ScheduledEventSource	The object representing the event source. See the following table for details.	VBScheduledEventSource
ScheduledEventDestRecord	The object representing the destination of the recorded content. See the following table for details.	VBScheduledEventDestRecord
ScheduledEventDestRecordVBrick	The object representing all destination record for the VBrick (VBStar). See the following tables for details.	VBScheduledEventDestRecordVBrick

The following fields must be set on the **VBScheduledEventSource** object prior to assigning it to the ScheduledEventSource field of the scheduledEventObj object.

Name	Description	Type
ContentID	The content ID of the original content to rebroadcast.	int
ContentInfo	Set to the object returned from ScheduleEventContentInfoGet.	VBScheduledEventContentInfo
EnumContentTypeID	The content type to record. This is always set to the value of the CONTENTTYPE .Live enum.	int
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventSourceID	The ID of the cource of the scheduled event. This is always set to zero.	int

The following fields must be set on the **VBScheduledEventDestRecord** object prior to assigning it to the ScheduledEventDestRecord field of the scheduledEventObj object.

Name	Description	Type
ScheduledEventID	The ID of the scheduled event. This is always set to zero.	int
ScheduledEventDestRecordID	The Id of the scheduled event destination record. This is always set to zero.	int
EnumScheduledEventDestRecordTypeID	The destination record type. Since this is not a recording but a playback operation, the SCHEDULEDEVENTDESTRECORDTYPE.None enum is used.	int

Name	Description	Type
DeviceVBrickSlotID	The ID of the VBrick slot that the original content is streaming out of. This is set to the value of the ContentSourceDeviceSlotInfo.DeviceSlotID field contained within the VBScheduledEventContentInfo object returned fromthe ScheduleEventContentInfoGet method.	int

The following fields must be set on the **VBScheduledEventDestRecordVBrick** object prior to assigning it to the ScheduledEventDestRecordVBrick field of the scheduledEventObj object:

Name	Description	Type
ScheduledEventDestRecordID	The ID of the scheduled event destination record. This is always set to zero.	int
ShouldFtpAfterRecord	Flag indicating whether or not to FTP the file into VEMS after the recording is complete. This is usually set to true.	boolean
DeviceVBrickSlotID	The ID of the VBrick slot that the original content is streaming out of. This is set to the value of the ContentSourceDeviceSlotInfo.DeviceSlotID field contained within the VBScheduledEventContentInfo object returned from the SchedulEventContentInfoGet method.	int

Output:

`VBVoidData object`

The **VBVoidData** object returned allows client code to determine if an error occurred during the ScheduleEventAdd operation by examining the **`Exception`** field.

Example Client Code:

```
//login
VBSession mySession = sll.UserLogin(Username, Password, ApplicationID, ClientIP,
UserLanguage);
if (mySession.Exception != null) throw new Exception("UserLogin failed");

//retrieve search criteria from GUI and retrieve the desired content with search
API call (not shown)
VBContent targetContent;

//get UTC time zone offset and daylight savings start/end months
int utcOffsetInMin = -300; //value for eastern United States
int daylightSvgsStartMth = 3; //starts in March
int daylightSvgsEndMth = 11; //ends in November

//1. get time zone info
VBIntData timeZoneData = sll.DateTimeGetBestGuessEnumTimeZoneID(utcOffsetInMin,
true, daylightSvgsStartMth, daylightSvgsEndMth, mySession.SessionID);

if (timeZoneData.Exception != null) throw new Exception("Could not get time zone
ID");
```

```
//2. retrieve schedule event content info
VBScheduledEventContentInfo eventContentInfo;
eventContentInfo = sll.ScheduleEventContentInfoGet(targetContent.ContentID,
mySession.SessionID);

if (eventContentInfo.Exception != null) throw new Exception("Could not get Content
Info");

//3. generate contentID for this recording
VBIntData recordingContentID =
sll.ScheduleEventContentSeedCreateFromContentWithMetadata(targetContent.ContentID,
mySession.SessionID);

if (recordingContentID.Exception != null) throw new Exception("Could not create
Content ID for recording");

//retrieve title and description content metadata from GUI (not shown here)
string recordTitle;
string recordDesc;

//create content object to represent the recording
VBContent newRecordedContent = new VBContent();
newRecordedContent.ContentID = targetContent.ContentID;
newRecordedContent.Title = recordTitle;
newRecordedContent.Description = recordDesc;

//4. update title and description
VBVoidData updateTitleDescResponse =
sll.ContentTitleDescriptionUpdate(newRecordedContent, mySession.SessionID);

if (updateTitleDescResponse.Exception != null) throw new Exception("Could not
update title and description of recorded content");


//retrieve name and description of the scheduled event from GUI (not shown here)
string eventName;
string eventDesc;

//construct the scheduled event object
VBScheduledEvent newEventToSchedule = new VBScheduledEvent();
newEventToSchedule.ScheduledEventID = 0; //always zero
newEventToSchedule.Description = eventDesc;
newEventToSchedule.Name = eventName;
newEventToSchedule.EnableRecurrence = false; //false if not a recurring event
newEventToSchedule.ShouldDeleteWhenExpired = false;
newEventToSchedule.EnumScheduledEventTypeID = (int)SCHEDULEEVENTTYPE.Record; //
always set to this          newEventToSchedule.EnumScheduledEventCreatedBySourceID
= (int)SCHEDULEDEVENTCREATEDBYSOURCE.Scheduler; //always set to this
newEventToSchedule.EnumTimeZoneID = timeZoneData.ReturnValue;
newEventToSchedule.IsTempEvent = false; //always false
newEventToSchedule.MetadataContentID = recordingContentID.ReturnValue; //content
ID for recording

//set start/end times and duration (set it to start now and last for one hour)
newEventToSchedule.StartDate = DateTime.Now;
newEventToSchedule.StartDateMonth = DateTime.Now.Month;
newEventToSchedule.StartDateDay = DateTime.Now.Day;
newEventToSchedule.StartDateYear = DateTime.Now.Year;
newEventToSchedule.StartTimeHour = DateTime.Now.Hour;
newEventToSchedule.StartTimeMinute = DateTime.Now.Minute;
newEventToSchedule.EndDate = DateTime.Now.AddHours(1);
newEventToSchedule.EndDateMonth = DateTime.Now.AddHours(1).Month;
newEventToSchedule.EndDateDay = DateTime.Now.AddHours(1).Day;
newEventToSchedule.EndDateYear = DateTime.Now.AddHours(1).Year;
newEventToSchedule.EndTimeHour = DateTime.Now.AddHours(1).Hour;
newEventToSchedule.EndTimeMinute = DateTime.Now.AddHours(1).Minute;
newEventToSchedule.RecurrenceScheduleTimeDurationSeconds = 3600; //duration of the
event (1 hour)

//build the Event Source object
newEventToSchedule.ScheduledEventSource = new VBScheduledEventSource();
newEventToSchedule.ScheduledEventSource.ContentID = targetContent.ContentID; //ID
of live stream
```

```
newEventToSchedule.ScheduledEventSource.ContentInfo = eventContentInfo;
newEventToSchedule.ScheduledEventSource.EnumContentTypeID=(int)CONTENTTYPE.Live;
newEventToSchedule.ScheduledEventSource.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventSource.ScheduledEventSourceID = 0; //always zero

//build Destination Record object
newEventToSchedule.ScheduledEventDestRecord = new VBScheduledEventDestRecord();
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventID = 0; //always zero
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordID = 0; //
always zero
newEventToSchedule.ScheduledEventDestRecord.EnumScheduledEventDestRecordTypeID =
(int)SCHEDULEDEVENTDESTRECORDTYPE.VBrick; //always set to this value when recording
to VBStar newEventToSchedule.ScheduledEventDestRecord.DeviceVBrickSlotID =
eventContentInfo.ContentSourceDeviceSlotInfo.DeviceSlotID;

//build Destination Record for the VBrick
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick = new
VBScheduledEventDestRecordVBrick();
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick.Schedul
edEventDestRecordID = 0; //always zero
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick.ShouldF
tpAfterRecord = true; //always true
newEventToSchedule.ScheduledEventDestRecord.ScheduledEventDestRecordVBrick.DeviceV
BrickSlotID = eventContentInfo.ContentSourceDeviceSlotInfo.DeviceSlotID;


//5. Add the event
VBScheduledEventAddUpdateDeleteReturn scheduledEventResponse;
scheduledEventResponse = sll.ScheduleEventAdd(newEventToSchedule,
mySession.SessionID);

if (scheduledEventResponse.Exception != null) throw new Exception("Scheduled
recording failed");

//logout
VBVoidData logOutResponse = sll.UserLogout(mySession.SessionID);
if (logOutResponse.Exception != null) throw new Exception("UserLogout failed");
```

Chapter 12

Embed and Share Video

Topics in this document

Embedding Code & Sharing a URL. 117
Embed Code 117
Share URL Code 120

Embedding Code & Sharing a URL

The VEMS Mystro API does *not* provide the ability to generate the HTML code needed to embed a video or share a video. VEMS Mystro currently uses client side JavaScript to generate this code. The code is not too complex and can be implemented in JavaScript, C#, or any other language of the developer's choice. The format of the Embed code will be explored first followed by the format of the Share code.

Embed Code

The Embed code follows 2 different templates; there is one for regular video content and one for video clip content. The template for regular video content is discussed followed by the template for video clip content.

Template 1:

Syntax:

```
        <iframe class='embedPlayer' type='text/html' src='<protocol><serverHost> /
VEMSWeb/Widgets/EmbedPlayerControllerWidget.html?contentID=<targetContentID>'
width='<width>' height='<height>' frameborder='0' />
```

Inputs:

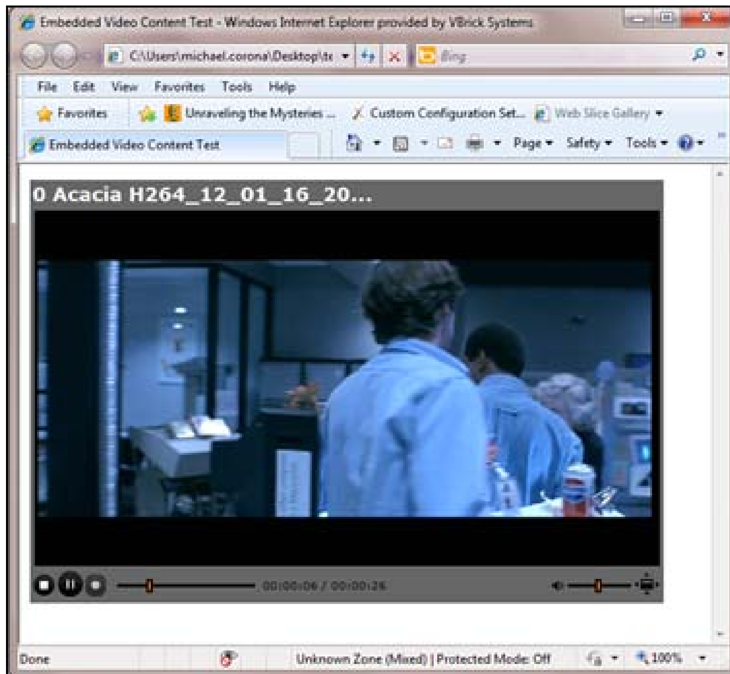
Name	Description	Type
<protocol>	The version of HTTP being used, either http:// or https://	string
<serverHost>	Either the IP address or DNS name of the VEMS Mystro server.	string
<targetContentID>	The content ID of the content to embed.	int
<width>	The width of the iframe.	int
<height>	The height of the iframe.	int

Example Client Code:

```
<iframe class='embedPlayer' type='text/html' src='http://172.22.2.217/VEMSWeb/
Widgets/EmbedPlayerControllerWidget.html?contentID=31' width='600' height='400'
frameborder='0' />
```

Now the developer can take the generated iframe element and embed it inside of another web page:

```
<html>
<head>
<title>Embedded Video Content Test</title>
</head>
<body>
<iframe class='embedPlayer' type='text/html' src='http://172.22.2.217/VEMSWeb/
Widgets/EmbedPlayerControllerWidget.html?contentID=31' width='600' height='400'
frameborder='0' />
</body>
</html>
```



Template 2:

Syntax:

```
    <iframe class='embedPlayer' type='text/html' src='<protocol><serverHost> /
VEMSWeb/Widgets/
EmbedPlayerControllerWidget.html?contentID=<targetContentID>&clipContentID=<clipCo
ntentID>' width='<width>' height='<height>' frameborder='0' />
```

Inputs:

Name	Description	Type
<protocol>	The version of HTTP being used, either http://or https://	string
<serverHost>	Either the IP address or DNS name of the VEMS Mystro server.	string
<targetContentID>	The content ID of the content to embed.	int

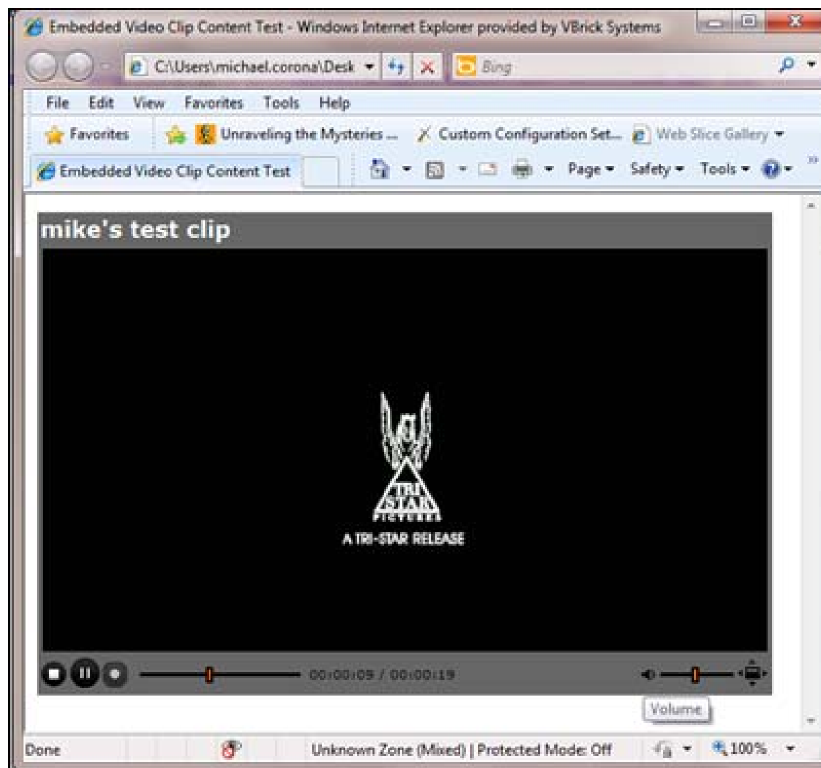
Name	Description	Type
<clipContentID>	The content ID of the video clip to embed.	int
<width>	The width of the iframe.	int
<height>	The height of the iframe.	int

Example Client Code:

```
<iframe class='embedPlayer' type='text/html' src='http://172.22.2.217/VEMSWeb/
Widgets/EmbedPlayerControllerWidget.html?contentID=31&clipContentID=25'
width='600' height='400' frameborder='0' />
```

Now the developer can take the generated iframe element and embed it inside of another web page:

```
<html>
<head>
<title>Embedded Video Clip Content Test</title>
</head>
<body>
<iframe class='embedPlayer' type='text/html' src='http://172.22.2.217/VEMSWeb/
Widgets/EmbedPlayerControllerWidget.html?contentID=31&clipContentID=25'
width='600' height='400' frameborder='0' />
</body>
</html>
```



Share URL Code

The Share code follows 2 different templates; there is one for regular video content and one for video clip content. The template for regular video content is discussed followed by the template for video clip content.

Template 1:

Syntax:

```
        <protocol><serverHost>/VEMSWeb/VEMSHost.html?VBTemplate=Templates/
VideoInfoTemplate.xml&contentID= <targetContentID>&shared=1
```

Inputs:

Name	Description	Type
<protocol>	The version of HTTP being used, either http://or https://	string
<serverHost>	Either the IP address or DNS name of the VEMS Mystro server.	string
<targetContentID>	The content ID of the content to share.	int

Example Client Code:

```
http://172.22.2.217/VEMSWeb/VEMSHost.html?VBTemplate=Templates/
VideoInfoTemplate.xml&contentID=787&shared=1
```

Template 2:

Syntax:

```
        <protocol><serverHost>/VEMSWeb/VEMSHost.html?VBTemplate=Templates/
ClipVideoInfoTemplate.xml&contentID=<targetContentID>&clipContentID=<clipContentID
>&shared=1;
```

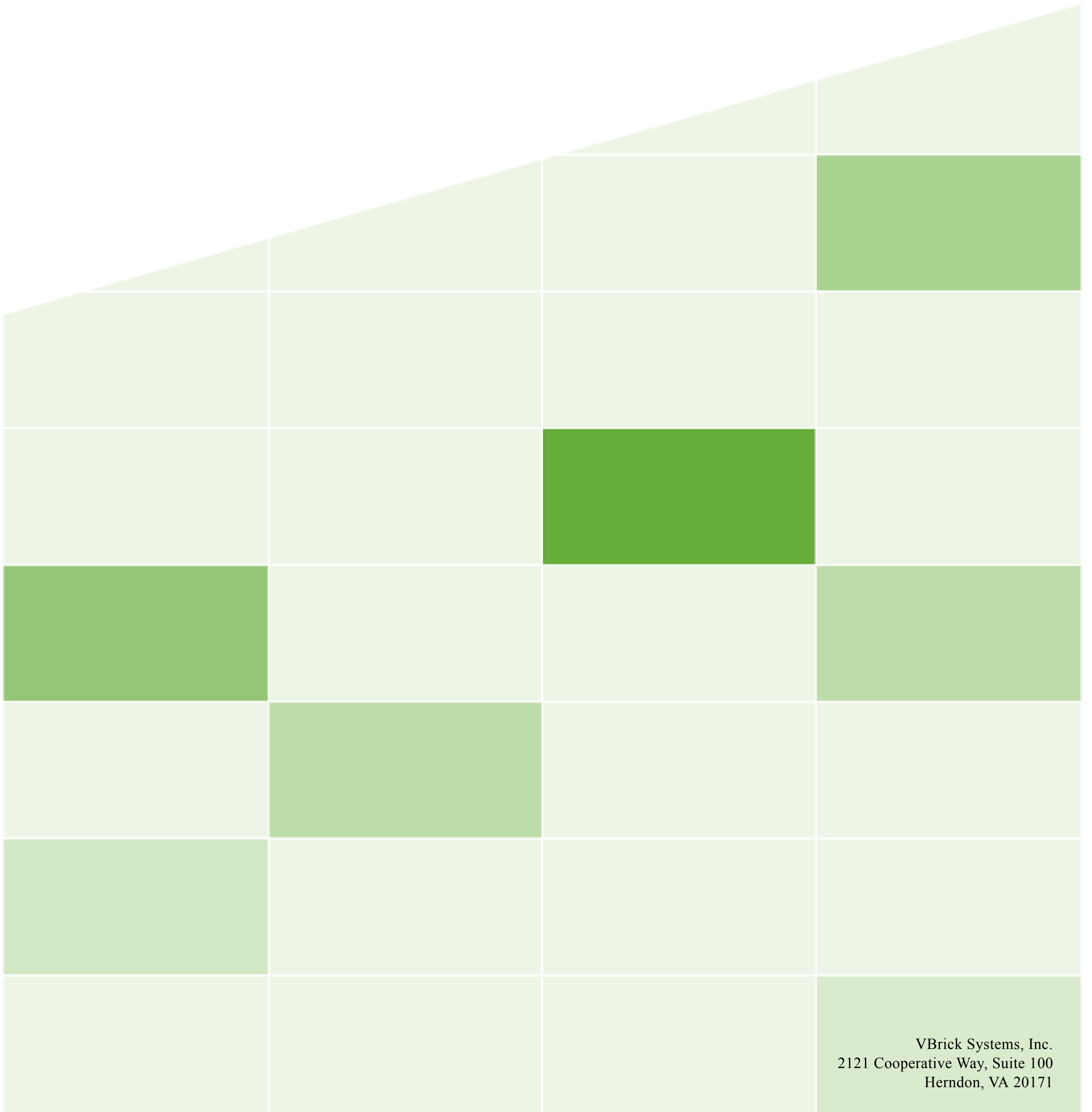
Inputs:

Name	Description	Type
<protocol>	The version of HTTP being used, either http://or https://	string
<serverHost>	Either the IP address or DNS name of the VEMS Mystro server.	string
<targetContentID>	The content ID of the content to share.	int
<clipContent>	The content ID of the video clip to share.	int

Example Client Code:

```
http://172.22.2.217/VEMSWeb/VEMSHost.html?VBTemplate=Templates/
ClipVideoInfoTemplate.xml&contentID=298&clipContentID=2447&shared=1
```

Note: Please observe that the difference between the two *Share* templates is the template file being used. The *regular* video content template uses VideoInfoTemplate.xml while the video clip template uses the ClipVideoInfoTemplate.xml.



VBrick Systems, Inc.
2121 Cooperative Way, Suite 100
Herndon, VA 20171