K2 AppServer API Developer’s Guide

Revisions

Version	Revision Date	Description
1.0	February 15, 2007	Initial Release
1.1	August 6, 2009	Fixed errors, updated error codes in Appendix E

Table of Contents:

1 Introduction

The purpose of this document is to explain the structure and concepts of the AppServer API and to give examples for common tasks. It is not intended to contain all information about the AppServer API, but rather to serve as a good primer for understanding the concepts and use.

For more API information developers should reference the *K2 API* online at:
http://www.gvgdevelopers.com/K2DevGuide/API/index.html

Additionally, you should visit the online K2 *AppServer Developer's Guide* for other examples, tutorials, and information:
http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html

This document is a work-in-progress. Some sections are marked "TBD" indicating that they still need "to be done". In those cases effort has been made to provide enough information in a related example that a developer could figure out what they need. One such case is where an example is written in one programming language, but the example still needs to be written in a second language.

Developers should visit the online *K2 AppServer Developer's Guide* to download the latest update to this document.

2 AppService API

2.1 Overview

The K2 Client supports 4 different APIs: AMP, BVW, VDCP, and AppServer. The first three APIs are protocol APIs (see the "*K2 Protocol Developer's Guide"* for more information). The AppServer API is the K2's native API. Applications like AppCenter use the AppServer API.

The AppServer API evolved from our work with previous Microsoft COM servers on the Grass Valley Profile, M-Series, and Turbo video servers. We used Microsoft's .NET technology to wrap and extend this functionality. This gives us the ability to remotely control the K2 using .NET remoting and interface with the K2 using managed and unmanaged code. Using the AppServer interface the K2 has been successfully controlled from C++, C#, and Python applications.

2.2 AppService objects

The AppService API is implemented by a service that runs on the K2 Client called "Grass Valley AppService". Its job is to provide clients with a way to control or retrieve information from the K2 Client. AppService is implemented by a singleton AppServerMgr object whose job is to create and manage AppServer objects for client applications. A client application first connects to the AppServerMgr, then asks the AppServerMgr to create an AppServer for it.

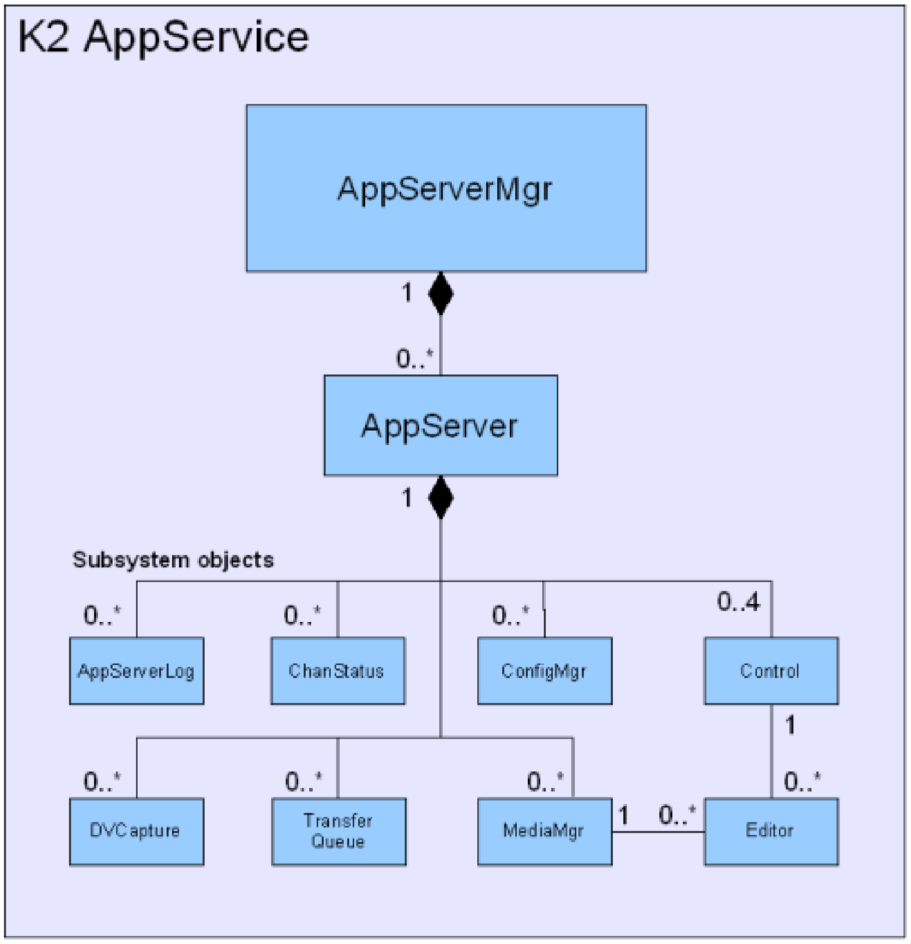


Figure 1 – AppService class diagram

2.3 AppServer and Subsystem objects

Once a client application has an AppServer object, the client can then ask the AppServer to create the Subsystem objects it needs to perform specific functionality. Here is a list of Subsystem objects and the type functionality each subsystem performs:

Subsystem	**Functionality**
AppServerLog	writing messages to the K2's log
ChanStatus	getting the K2's channel status
ConfigMgr	getting the K2's configuration
Control	controlling the K2's players and recorders
Editor	editing the K2's assets
MediaMgr	managing the K2's volumes, bins, and assets
Status	getting the K2's status
Transfer	transferring, importing, and exporting K2 assets

2.4 Client calling sequence

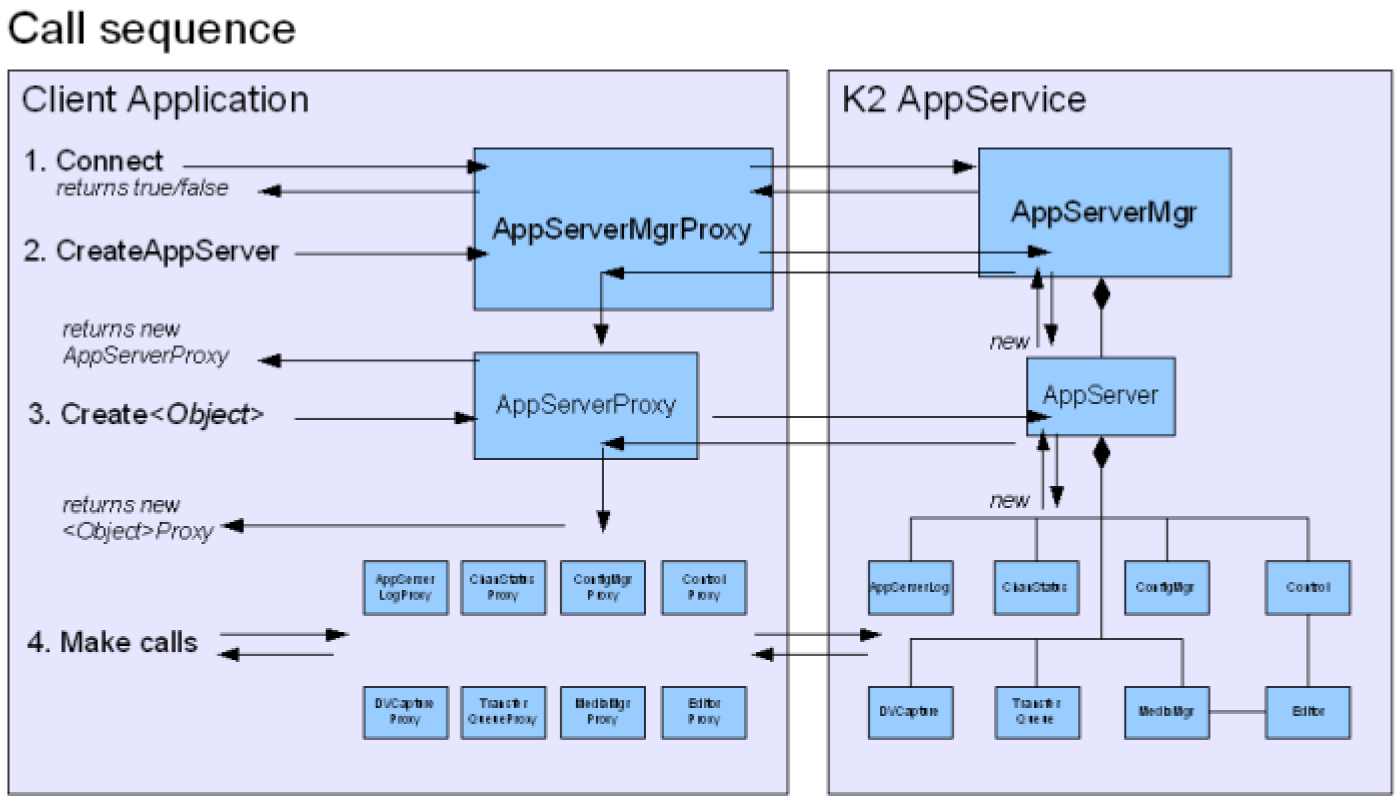


Figure 2 – Client Calling Sequence

Proxies:
Client applications interact with the K2 via "proxy" objects. These are objects in the client application that represent the real objects in the K2. The proxy object's job is to simplify calls and to encapsulate house-keeping tasks so that the client application does not need to worry about it.

Here are the basic steps for a client application to connect to a K2 and do work:

1. Connect to a K2's AppService:
1a. Create an AppServerMgrProxy object in the client application.
1b. Tell it which K2 Client you want to connect to by calling the "SetHost" function.
1c. Pass in user credentials by calling the "SetUserCredentials" function.
1d. Call "Connect()" function. It returns true or false.

2. Create an AppServer:
Once connected, create an AppServer on the K2 by calling "CreateAppServer". This will create an AppServer on the K2 and return an AppServerProxy object back to the client application.

3. Create subsystem objects:
Now that you have an AppServer for your client, create the subsystem object(s) that you need by calling "Create<Object>" (i.e. CreateController, CreateLog, etc.). This will create the object on the K2 and return an <Object>Proxy object to the client.

4. Make K2 calls:
Make calls on subsystem proxy objects to control or retrieve information from the K2.

5. Cleanup:
When finished, cleanup and delete the AppServer and subsystem objects by calling CloseConnection on the AppServerProxy.

2.5 AppServer and suite naming

Suites and Security:
When a client requests an AppServer, it passes in a unique suite name and a user name. The concept is similar to someone using an editing suite. While that person is in the suite, they have access to all of their own resources. Similarly, each AppServer for a user and suite has its own resources. This is also what tells AppService whether-or-not to create a new AppServer. If two client applications request an AppServer with the same suite name and user name, then both applications will get back the same AppServer object.

2.6 Closing versus Suspending an AppServer

AppService is designed so that when a client application exits it can either:

1) Shutdown all of the resources it acquired, or
2) Leave the resources up-and-running so that it can reacquire the same resources the next time the application runs

One use for this would be an application that keeps a player or recorder running even when the client application is not running. This feature is implemented by calling the CloseConnection or SuspendConnection functions on the AppServer object.

⚠ **Applications are responsible for managing their resources.**

If an application exits and leaves a recorder suspended in a recording state, the recorder will continue to use disk space until it either uses up the rest of the drive space or until the application reconnects and tells it to stop.

Note: AppServers will not remain suspended if the AppService is restarted, the K2 is rebooted, or another application steals all of the AppServer's resources.

2.7 Channel stealing

Channels on the K2 Client are shared resources. If a client is using a channel and another client requests to use that same channel, the channel will be stolen away from the first client and given to the second client. If it is important to prohibit this, the application should first check to see if the channel is owned by another application first (see 4.4.1 Channel owner).

2.8 K2 Terminology

2.8.1 Volume

A volume is a set of media drives that functions as a single physical disk. These drives may be contained with the K2 (i.e. in a standalone K2 Client) or externally (i.e. disk array attached to a K2 Server on a SAN).

The K2 system uses 'V:' for the volume name. (Note that the K2 Emulator uses drive letter 'C:' instead).

2.8.2 Bin

A bin is a container in the volume used to organize assets in the same way as directories or folders are used on a typical computer system. A bin is associated with a single disk volume. On the K2 Client there is always a "V:\default" bin and a "V:\Recycle Bin" bin

2.8.3 Asset

An asset is a media-related entity in the database. This is a general term that encompasses clips, movies, playlists, programs, audio-only clips, etc. For example, "V:\default\Clip" is an asset on the system.

2.8.4 Movie

When you record on a K2 each video, audio, and timecode track is recorded to separate files. In the database all of these files are associated together as a movie. In the user interface and API the movie is presented as a single entity such as "V:\default\Clip".

2.8.4.1 Simple Movie

A simple movie is a movie that consists of a single segment for each media file. This is the case when you start recording a new movie then later press stop. All media in this movie was created in a single "take" making this a simple movie. Adjusting marks does not change the simple status either.

2.8.4.2 Complex Movie

A complex movie is movie that has multiple segments. This occurs if you record in multiple takes or if the movie has had media inserted into or deleted from it.

2.8.5 Clip

A clip is technically a simple movie, but often the term "clip" is used synonymously as an asset.

2.8.6 Playlist

A playlist is a complex movie that has additional metadata stored with it. This metadata includes section and edit information. This metadata is used by the Playlist application to provide additional functionality at the section and edit transition points. This functionality includes looping, pausing at the end of an edit or section, pausing at the start of the next edit or section, etc. Playlists may also be played in the regular Player/Recorder applications, but the section and edit pause/loop functionality will be ignored.

2.8.7 Program

A playlist may be saved as a program. A program is essentially a playlist with all of the section and edit metadata removed from it.

2.8.8 Universal Resource Identifiers (URIs)

Universial Resource Identifiers are used for specifiying asset, bin, and volume names in the API. URIs on the K2 are defined as follows:

```
    edl/cmf//<host>/<volume>/<bin>/<asset>
```

`"edl"` stands for "edit decision list"
`"cmf"` stands for "common movie format"
`<host>` is the K2's host name or IP address (e.g. "K2-123")
`<volume>` is the volume or drive letter (e.g. "V:")
`<bin>` is the bin or directory name (e.g. "default")
`<asset>` is the asset's name (e.g. "Clip")

Note: the "edl" and "cmf" designators are legacy designators from the Profile days.

An example URI:

```
edl/cmf//K2-123/V:/default/Clip
```

2.8.9 Uniform Media Locators (UMLs)

Uniform Media Locators are used for specifiying transfer source and destination locations (see 8.1Transfer call syntax). The format for UMLs is:

```
<host>/<type>/<filename>
```

Where:

`<host>` is the K2's host name or IP address (e.g. "K2-123")
`<type>` is a file type (e.g. "explodedFile", "gxfFile", "mxfFile", "movFile", "aviFile", "mpgFile")
`<filename>` is the asset's fully pathed name including volume, bin, and asset (e.g. "V:/default/Clip" or "C:/temp/foo")

2.9 Getting started

The AppServer API use .NET Remoting for communicating with the Grass Valley AppService running on the K2 Client. Because of this applications can be written to run directly on the K2 Client or on a remote PC. This is how AppCenter runs either on the K2 Client or remotely on a Control Point PC.

2.9.1 Working directly on a K2

You can setup your development environment directly on the K2. After that most dependencies (DLLs, etc.) can be found in the C:\profile directory.

2.9.2 Working on a remote PC

To work from a remote PC, you will need to install the Control Point software that closely matches the version of software that the K2 Client is running. To find this:

1. Go to http://www.grassvalley.com.
2. On the left, click "Services & Support"
3. In the middle-top, click "Software" then below click "Broadcast Products Software"
4. In the middle-top, click "Servers/Media Storage"
5. Below click "K2 Media Server & Media Client".
6. Click the closest matching version number of "K2 Media Client Software".
7. Scroll down and download the "Control Point Installation Package".

Install Control Point on your PC. You will now have everything registered that you need. Additionally, you will have a remote version of AppCenter installed. Test that you can connect to your K2 Client by running AppCenter.

2.9.3 Working with the K2 Emulator

If you do not have a K2 Client to connect to you can use the K2 Emulator for testing. The PC requirements for the Emulator are the same as the Control Point PC (see 10.6 ControlPoint PC).

Please visit the following link to learn more about its functionality & limitations and installation & configuration steps:

K2 Emulator
http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BK2%20Emulator%5D%5D

2.9.4 Additional includes for C++ projects

If you are writing an unmanaged C++ application you will need to download the TLH include files from this link:

http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BInstall%20Client%20Libraries%5D%5D

You can also download the sample project C++ project below and see how it uses them.

2.9.5 Sample projects

An unmanaged C++ project and a managed C# project are available for download here:

Sample Projects
http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BSample%20Projects%5D%5D

These projects were written with Visual Studio 2003. K2 client applications can be built using Visual Studio 6 and Visual Studio 2005 as well.

2.9.6 Build location and dependencies

In order to test your application you will need to run it in the C:\profile directory. This is so that it can find the AppServer client DLLs. While you are developing it is a good idea to set your build output to C:\profile and run it from there.

3 Connections

3.1 Connecting to a K2 Client

To connect to a K2 Client you:
1. Create an AppServerMgrProxy object in your client application.
2. Tell it which K2 Client you want to connect to by calling the "SetHost" function.
3. Pass in user credentials by calling the "SetUserCredentials" function.
4. Then call "Connect()" function. It returns true or false.

C# Code

```
// create an AppServerMgr proxy object
AppServerMgrProxy appServerMgrProxy = new
AppServerMgrProxy();

// set the host name of the K2 server
string host = "K2machinename";
appServerMgrProxy.SetHost(host);

// set the user credentials
string username = "username";
string password = "password";
string domain = "domain";
appServerMgrProxy.SetUserCredentials(username,
password, domain, false);

// connect to the AppServerMgr
if ( !appServerMgrProxy.Connect() )
  Console.WriteLine("ERROR: Could not connect to
    AppService on " + host + ".");
else
  Console.WriteLine("Successfully connected to
    AppService.");

// ... do some work ...

// disconnect
appServerMgrProxy.Disconnect();
```

C++ Code

```
// Initialize the COM library
CoInitialize(NULL);
HRESULT hr;

// create an AppServerMgr proxy object
IAppServerMgrProxyPtr spAppServerMgrProxy
(__uuidof(AppServerMgrProxy));

// set the host name of the K2 server
_bstr_t bstrHost("K2machinename");
hr = spAppServerMgrProxy->SetHost(bstrHost);

// set the user credentials
_bstr_t bstrUsername("username");
_bstr_t bstrPassword("password");
_bstr_t bstrDomain("domain");
VARIANT_BOOL encrypted(FALSE);
hr = spAppServerMgrProxy->SetUserCredentials(
  bstrUsername, bstrPassword, bstrDomain,
  encrypted);

// connect to the AppServerMgr
VARIANT_BOOL retVal;
hr = spAppServerMgrProxy->Connect(&retVal);
if ( !retVal )
  printf("ERROR - couldn't connect to the
    AppServerMgr\n");

// ... do some work ...

// disconnect
spAppServerMgrProxy->Disconnect(&retVal);
```

3.2 Creating an AppServer object

Once connected, create an AppServer object on the K2 by calling the AppServerMgrProxy's "CreateAppServer" function. This will create an AppServer object on the K2 and return an AppServerProxy object back to the client application.

C# Code

```
// create a suite name. make this name unique unless
// you want to share an AppServer object between
// applications
string suiteName = "TestSuite";

// set the application's name. this is used in
// logging.
string appName = "TestApp";

// create a new AppServer object. If a new
// AppServer was created newConnection will
// return with the value true. If a previously
// created AppServer is being returned (i.e.
// a suspended AppServer), newConnection
// will return with the value false.
bool newConnection = false;
IAppServer iappServer =
    appServerMgrProxy.CreateAppServer(suiteName,
      appName, out newConnection);
```

C++ Code

```
// create a suite name. make this name unique unless
// you want to share an AppServer object between
// applications
_bstr_t bstrSuiteName("TestSuite");

// set the application's name. this is used in
// logging.
_bstr_t bstrAppName("TestApp");

// create a new AppServer object. If a new
// AppServer was created newConnection will
// return with the value true. If a previously
// created AppServer is being returned (i.e.
// a suspended AppServer), newConnection
// will return with the value false.
short nNewConnection;
```

```
IAppServerPtr spAppServer;
HRESULT hr = spAppServerMgrProxy->
CreateAppServer(bstrSuiteName,
bstrAppName, &nNewConnection, &spAppServer);
```

3.3 Closing an AppServer object

When finished with an AppServer and its resources (see 2.6 Closing versus Suspending an AppServer) call "CloseConnection" on the AppServerProxy.

C# Code

```
// close the AppServer object and all of
// its resources
iappServer.CloseConnection();
```

C++ Code

```
// close the AppServer object and all of
// its resources
spAppServer->CloseConnection();
```

3.4 Suspending an AppServer object

If you want to stop your application, but leave your K2 resources up-and-running (see 2.6 Closing versus Suspending an AppServer) call "SuspendConnection" on the AppServerProxy.

C# Code

```
// suspend the AppServer object and
// all of its resources
iappServer. SuspendConnection ();
```

C++ Code

```
// suspend the AppServer object and
// all of its resources
spAppServer-> SuspendConnection ();
```

3.5 Reacquiring an AppServer object

To reacquire a previously suspended AppServer, call AppServerMgrProxy's CreateAppServer method and pass in the suitename. You will received back the previous AppServerProxy object and newConnection's value will be false.

4 Getting system information

4.1 System status

Below is sample code for checking out the status of the K2. If the K2 is up-and-running and it has finished its startup sequence, you should see the following values returned:

SystemMode: Normal BootStage: UP_AND_RUNNING BootCode: NORMAL

Check the Status namespace for a list of other BootStage and BootCode values at:
http://www.gvgdevelopers.com/K2DevGuide/API/GrassValley.Mseries.Status.html

C# Code

```
string systemMode;
BootStage bootStage;
BootCode bootCode;

// get status of the K2 Client
appServerMgrProxy.GetSystemStatus(out systemMode,
    out bootStage, out bootCode);
Console.WriteLine("SystemMode = {0} BootStage = {1}
    BootCode = {2}", systemMode, bootStage,
    bootCode);
```

C++ Code

```
BSTR systemMode;
BootStage bootStage;
BootCode bootCode;

// get status of the K2 Client
spAppServerMgrProxy->GetSystemStatus(&systemMode,
    &bootStage, &bootCode);
CString sSystemMode = systemMode;
printf("SystemMode: %s BootStage: %d BootCode:
    %d\n\n", sSystemMode, bootStage, bootCode);
```

4.2 Software version

The code below will print out the version number of the software running on the K2 Client:

C# Code

```
string version
appServerMgrProxy.GetMSeriesIdInfo(
    IdComp.SW_VER, out version);
Console.WriteLine("K2 Software Version: " +
  version);
```

C++ Code

```
BSTR version;
spAppServerMgrProxy->GetMSeriesIdInfo(
    IdComp_SW_VER, &version);
CString sVersion = version;
printf("K2 Software Version: %s\n", sVersion);
```

4.3 Configuration information

4.3.1 Getting a K2 Client's configuration (config.xml)

C# Code

```
// create a ConfigMgr object
IConfigMgr configMgr = iappServer.CreateConfigMgr(appName);

// get config XML data (i.e. c:\profile\config\config.xml)
string xmlData;
configMgr.GetConfigXml(out xmlData);

// cleanup
configMgr.Dispose();
```

C++ Code

TBD

4.3.2 Parse channel information from the configuration data

C# Code

```
using System.Xml;

// setup a native XmlDocument to find the channel numbers and labels
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml(xmlConfig);

// need a namespace manager to parse the xml doc
XmlNamespaceManager namespaceManager = new XmlNamespaceManager(xmlDoc.NameTable);
namespaceManager.AddNamespace("ns", "x-schema:#Schema1");

// get all channels and loop thru them to find their index and label
XmlNodeList channelList = xmlDoc.SelectNodes(("./Config/Channel"), namespaceManager);
if (channelList.Count > 0)
{
    IEnumerator channelNodeEnum = channelList.GetEnumerator();
    bool bMoreChannels = channelNodeEnum.MoveNext();

    // loop while there are more channels
    do
    {
        // get the channel node. get the "id" and "channelIndex" values
        XmlNode channelNode = (XmlNode)channelNodeEnum.Current;
        string channelName = channelNode.Attributes.GetNamedItem("id").Value;
        int channelNumber = Int32.Parse(channelNode.Attributes.GetNamedItem("channelIndex").Value);

        bMoreChannels = channelNodeEnum.MoveNext();
    } while (bMoreChannels);
}
```

C++ Code

TBD

4.4 Channel properties

4.4.1 Channel owner

The code below displays information about the owner of a K2 Client's channel. The information returned includes the machine, the suite name, the user's name, and application that owns the channel.

C# Code

```
// use channels C1 - C4 on SD systems
// use channels P1 - P4 & R1 - R2 on HD systems
string channel = "C1";
string machine = "";
string suite = "";
string user = "";
string app = "";

appServerMgrProxy.GetChannelOwnerInfo(channel,
  out machine, out suite, out user, out app);

Console.WriteLine("machine = {0} suite = {1} user =
{2} app = {3}", machine, suite, user, app);
```

C++ Code

```
// use channels C1 - C4 on SD systems
// use channels P1 - P4 & R1 - R2 on HD systems
_variant_t varChan("C1");
BSTR bstrMachine;
BSTR bstrSuite;
BSTR bstrUser;
BSTR bstrApp;

spAppServerMgrProxy->GetChannelOwnerInfo(varChan,
  &bstrMachine, &bstrSuite, &bstrUser, &bstrApp);

CString sMachine = bstrMachine;
CString sSuite = bstrSuite;
CString sUser = bstrUser;
CString sApp = bstrApp;

printf("machine = %s suite = %s user = %s app = %s\n",
sMachine, sSuite, sUser, sApp);
```

4.4.2 Channel properties

Channel status is provided for each K2 channel primarily by channel objects which control hardware resources and timeline attributes. Channel status can be accessed by various Channel Status properties.

If you need to get multiple properties it is more efficient to make one call for all properties rather than multiple calls for each property.

C# Code

```
// create a ChanStatus object
string appName = "TestApp";
string channel = "C1";
IChanStatus ichanStatus =
iappServer.CreateChanStatus(appName, channel);

// get a single property
string timeremain = (string)
ichanStatus.GetStatus("timeremainingstr");
Console.WriteLine("timeremain = " + timeremain);

// get multiple properties
string query = "ownerName+assetname+videoFormatStr";
object[] results = (object[])
ichanStatus.GetStatus(query);
foreach (object obj in results)
{
 Console.WriteLine("obj = " + obj);
}

// cleanup
ichanStatus.Dispose();
```

C++ Code

```
// create a ChanStatus object
_bstr_t bstrAppName("TestApp");
_bstr_t bstrChannel("C1");
IChanStatusPtr spChanStatus;
HRESULT hr = spAppServer->CreateChanStatus(
_bstrAppName, bstrChannel, &spChanStatus);

// get a single property
VARIANT varResult;
hr = spChanStatus->
  GetStatus(_bstr_t("timeremainingstr"),
  &varResult);

CString sTimeRemaining = varResult.bstrVal;
printf("TimeRemaining: %s\n", sTimeRemaining);

// get multiple properties
hr = spChanStatus->
GetStatus(_bstr_t("ownerName+assetname+videoFormatStr"),
&varResult);

SAFEARRAY *pArray = varResult.parray;
VARIANT *pVar = NULL;
long arrIndex[1];

// get the first property
arrIndex[0] = 0;
SafeArrayPtrOfIndex(pArray, arrIndex,
  (void **)&pVar);
String sOwnerName = pVar->bstrVal;

// get the second property
arrIndex[0] = 1;
SafeArrayPtrOfIndex(pArray, arrIndex,
  (void **)&pVar);
CString sAssetName = pVar->bstrVal;

// get the third property
arrIndex[0] = 2;
SafeArrayPtrOfIndex(pArray, arrIndex,
  (void **)&pVar);
CString sVideoFormat = pVar->bstrVal;

printf("OwnerName: %s AssetName: %s VideoFormat:
%s\n\n", sOwnerName, sAssetName, sVideoFormat);
```

4.4.3 Change protocol listener

Starting in K2 software version 3.2 each channel will always have AppServer API control as well as one protocol controller. Which protocol controls can be configured in AppCenter's System Configuration panel. Additionally, the protocol can be configured programmatically as follows:

C# Code

```
// set a channel's protocol listener to AMP
Controller.SetChannelProperty("protocol", "AMP");

// make sure you give the system time to close previous listener & start new listener!
```

```
Sleep(2000);
```

C++ Code

```
// set a channel's protocol listener to AMP
HRESULT hr = spController->SetChannelProperty( _bstr_t("protocol"), _variant_t("AMP") );

// make sure you give the system time to close previous listener & start new listener!
Sleep(2000);
```

4.5 File system information

4.5.1 Space remaining

C# Code

```
// create a chanstatus object
IChanStatus chanStatus = iappServer.CreateChanStatus(appName, channel);

// get multiple properties
object[] results = (object[])
chanStatus.GetStatus("timeremaining+timeremainingstr+totalstorage+remainingstorage+pcntstorageremaining");
foreach (object obj in results)
{
    Console.WriteLine("objtype = " + obj.GetType() + " obj = " + obj);
}

// cleanup
chanStatus.Dispose();
```

C++ Code

5 Players, Recorders, and Transport Control

5.1 Load, play, and eject a clip

C# Code

```
bool isNewController = false;

// create a controller
ISimpleController icontroller = iappServer.CreateController(
"YourApplicationName", "C1", out isNewController);

// cast it to a player recorder and load a clip
ISimplePlayerRecorder player = (ISimplePlayerRecorder) icontroller;
player.Load("edl/cmf//local/V:/default/Clip");

// play for 3 seconds
player.Play();
Thread.Sleep(3000);

// eject the clip
player.Eject();

// close the channel
icontroller.CloseChannel();
```

C++ Code

```
short nIsNewController;
ISimpleControllerPtr spController;

// create a controller
HRESULT hr = spAppServer->CreateController(_bstr_t("YourApplicationName"), _bstr_t("C1"),
 &nIsNewController, &spController);

// cast it to a player recorder and load a clip
ISimplePlayerRecorderPtr spPlayer = (ISimplePlayerRecorderPtr) spController;
hr = spPlayer->Load(_bstr_t("edl/cmf//local/V:/default/Clip"));
```

```
// play for 3 seconds
hr = spPlayer->Play();
Sleep(3000);

// eject the clip
hr = spPlayer->Eject();

// close the channel
hr = spController->CloseChannel();
```

5.2 Record a clip

<u>C# Code</u>

```
bool isNewController = false;

// create a controller for channel C1
ISimpleController icontroller = iappServer.CreateController(
"YourApplicationName", "C1", out isNewController);

// cast the controller to an ISimplePlayerRecorder interface.
ISimplePlayerRecorder recorder = (ISimplePlayerRecorder) icontroller;

// generate a unique clip name in the V:\default bin
recorder.SetCurrentBin("V:", "default");
string newclip = recorder.GenerateUniqueName("test", "_");

// create the new clip and record
recorder.New(newclip);
recorder.Record();

// wait awhile...
Thread.Sleep(10000);

// stop, eject clip, and close channel
Recorder.Stop();
recorder.Eject();
icontroller.CloseChannel();
```

<u>C++ Code</u>

```
short nIsNewController;
ISimpleControllerPtr spController;
_variant_t varChanNum(m_bstrChannel);

// create the controller
HRESULT hr = spAppServer->CreateController(m_bstrAppName, varChanNum,
 &nIsNewController, &spController);

// generate a new clip name in the V:\default bin
spController->SetCurrentBin(_bstr_t("V:"), _bstr_t("default"));
BSTR bstrNewClip;
hr = spController->GenerateUniqueName(_bstr_t("test"), _bstr_t("_"), &bstrNewClip);

// cast the controller to a recorder
ISimplePlayerRecorderPtr spRecorder = (ISimplePlayerRecorderPtr) spController;

// create the new clip, cue the record, and set the record length to 10 seconds
spRecorder->New(bstrNewClip);
spRecorder->Record();

// wait awhile...
Sleep(10000);

// stop, eject clip, and close channel
hr = spRecorder->Stop();
hr = spRecorder->Eject();
hr = spController->CloseChannel();
```

5.3 Recording with a set duration

C# Code
```
// create the new clip, cue to beginning (field 0),
// set to record a 5 second clip, then record
recorder.New(newclip);
recorder.CueRecord(0);
recorder.SetRecordLength("00:00:05.00");
recorder.Record();
```

C++ Code
```
// create the new clip, cue to beginning (field 0),
// set to record a 5 second clip, then record
spRecorder->New(bstrNewClip);
spRecorder->CueRecord(0);
recorder.SetRecordLength(_bstr_t("00:00:05.00"));
spRecorder->Record();
```

5.4 Scheduled play and record

Often you want to synchronize the playout or recording of a channel or multiple channels at a specific time. You can schedule events to happen at specific triggers using context changes. Context changes are scheduled by issuing:

1. a “BeginContextChange” command, followed by
2. event(s) you want to happen, followed by
3. a “ScheduleContextChange” command specifying the trigger.

The code below illustrates how to start playing a clip at a specific timecode. Note: you can specify what the source of “timeofday” clock is for the K2 Client from AppCenter’s System Configuration panel.

C# Code
```
// create a controller
bool isNewController = false;
ISimpleController icontroller = connection.AppServer.CreateController(
    "YourAppName", "C1", out isNewController);

// cast it to a player recorder and load a clip
ISimplePlayerRecorder player = (ISimplePlayerRecorder) icontroller;
player.Load("edl/cmf//local/V:/default/Clip");

// must cue clip before you can schedule it!
player.CueStart();

// begin context change, telling it to play at a particular timecode
player.BeginContextChange();
player.Play();
player.ScheduleContextChange("timeofday", "09:00:00.00");

// to cancel schedule change
// player.CancelScheduledChange("timeofday", "09:00:00.00");

// wait awhile...
Thread.Sleep(45000);

// stop and eject the clip
player.Stop();
player.Eject();

// close the channel
icontroller.CloseChannel();
```

C++ Code
TBD

5.5 Start and End Limits

Start and end limits can be set on controllers. These functions restrict how far fast forward and rewind will go. They also set the points where CueStart() and CueEnd() functions will cue. Note that these limits are from the start and end of the clips unless marks are set. If marks are set, limits are in relation to the marks.

For example, if you have a clip with no marks and you set a start limit of 2 seconds, the function CueStart() will cue the clip at the 2 second mark. If the clip has mark in set to 3 seconds into the clip, CueStart() will cue to 5 seconds into the clip (mark in + start limit).

C# Code
```
// set start and end limits
player.SetStartLimit(300); // 5 seconds in fields
player.SetEndLimit(600); // 10 seconds in fields
```

C++ Code
```
_variant_t startLimit((long)300); // 5 seconds in fields
_variant_t endLimit((long)600); // 10 seconds in fields

// set start and end limits
HRESULT hr = spEditor->SetStartLimit(startLimit);
HRESULT hr = spEditor->SetEndLimit(endLimit);
```

5.6 Get timecode and using a timecode object

The K2 software includes a timecode class called VdrTimecode. The example below shows how you can use it. This class is useful specifying times for scheduling context changes (see 5.4 Scheduled play and record).

C# Code
```
using Interop.TimecodeLib;

string appName = "YourAppName";
string channel = "C1";

// create a controller
bool isNewController = false;
ISimpleController icontroller = connection.AppServer.CreateController(
    appName, channel, out isNewController);

// cast it to a player recorder and load a clip
ISimplePlayerRecorder player = (ISimplePlayerRecorder) icontroller;

// get video format of channel, get current time of channel
string videoformatstr = (string) player.GetStatus("videoformatstr");
int currentTimecode = (int) player.GetStatus("timecodeStr");

// create a new timecode object for video format
VdrTimecode timecode = new VdrTimecode();
timecode.VideoFormat = videoformatstr;

// set the value equal to the time
timecode.TcMask = (uint) currentTimecode;
Console.WriteLine("now = " + timecode.AsString);

// add 5 seconds to the timecode value
Console.WriteLine("add 5 seconds");
timecode.Add("00:00:05,00");

Console.WriteLine("later = " + timecode.AsString);

// close the channel
icontroller.CloseChannel();
```

C++ Code
TBD

5.7 Other transport functions

See the **K2 API** under GrassValley.MSeries.Control, ISimplePlayerRecorder interface methods for function syntax:
http://www.gvgdevelopers.com/K2DevGuide/API/index.html

Other transport functions include these ISimplePlayerRecorder functions:

Function	Functionality

Load	Loads an asset
New	Creates a new asset
Eject	Ejects an assets
Stop	Stops playing or recording
Play	Plays an asset
Record	Records an asset
FastFoward	Fast forwards (16x play rate)
Rewind	Rewinds (-16x play rate)
ShuttlePlay	Variable speed play
CueStart	Sets playback position to start
CueEnd	Sets playback position to end
Cue	Sets playback position to a specific timecode

5.8 Check a Controller's status

A channel's port is one of these states: idle, cued for play, playing, cued for recording, or recording. The property "shuttleSpeed" designates the rate the channel plays. It's range is -16.0 to 16.0. Fast forward and rewind are a combination of portState = Play and a "shuttleSpeed" property not equal to 1.0 or -1.0.

C# Code
```
// Get ChanStatus object
string appName = "Your Application Name";
string channel = "C1";
IChanStatus chanStatus = iappServer.CreateChanStatus(appName, channel);

// Get portState 0=idle, 1=cued for play
// 2=play, 3=cued for record, 4=recording
int portState = (int) ichanStatus.GetStatus("portState");

// cleanup
chanStatus.Dispose();
```

C++ Code
```
// get ChanStatus object
IChanStatusPtr spChanStatus;
HRESULT hr = spAppServer->CreateChanStatus(_bstr_t("YourApplicationName"),
 _bstr_t("C1"), &spChanStatus);

// Get portState 0=idle, 1=cued for play
// 2=play, 3=cued for record, 4=recording
VARIANT varResult;
hr = spChanStatus->GetStatus(_bstr_t("portState"), &varResult);
int nPortState = varResult.iVal;
```

6 Editing

6.1 Overview

The first thing you do to edit an asset on the K2 is get an Editor object for the asset. Editor objects are used to set mark in and mark out values, create subclips and playlists, stripe timecode, adjust audio gain, etc. Check out the K2 API, GrassValley.MSeries.Editor class, methods, and properties for more information.

http://www.gvgdevelopers.com/K2DevGuide/API/index.html

6.2 Get editor for an asset loaded on a channel

To get an editor from a channel, you load the clip in a controller and then ask the controller for an editor object.

⚠ **Be sure to call Detach() on an editor object when finished.**

Each editor holds a reference to an asset. If you don't detach the editor some asset operations may fail. For example, if you try to delete an asset, but still have an editor hanging on to the asset, the delete function will fail (message will be logged in the K2's log).

C# Code

```
// create a controller
string appName = "YourApplicationName";
string channel = "C1";
bool isNewController = false;
ISimpleController icontroller = iappServer.CreateController(
appName, channel, out isNewController);

// load a clip
ISimplePlayerRecorder player = (ISimplePlayerRecorder) icontroller;
player.Load("edl/cmf//local/V:/default/Clip");

// get an editor from the player
ISimpleEditor editor = player.GetEditor();

// do work

// you must detach your editor when you are done. If you don't you
// will hang onto a reference to the asset and some operations
// will fail (e.g. DeleteAsset).
editor.Detach();

// cleanup
editor.Dispose();
icontroller.CloseConnection();
```

C++ Code

```
// create a controller
short nIsNewController;
ISimpleControllerPtr spController;
HRESULT hr = spAppServer->CreateController(_bstr_t("YourApplicationName"), _bstr_t("C1"),
 &nIsNewController, &spController);

// cast it to a player recorder and load a clip
ISimplePlayerRecorderPtr spPlayer = (ISimplePlayerRecorderPtr) spController;
hr = spPlayer->Load(_bstr_t("edl/cmf//local/V:/default/Clip"));

// get an editor
ISimpleEditorPtr spEditor;
hr = spPlayer->GetEditor(&spEditor);

// do work

// you must detach your editor when you are done. If you don't you
// will hang onto a reference to the asset and some operations
// will fail (e.g. DeleteAsset).
hr = spEditor->Detach();
```

6.3 Get editor for an asset not loaded on a channel

To get an editor for a clip not loaded on a channel, you create MediaMgr object and ask it to create an editor for a clip.

⚠ **Be sure to call Detach() on an editor object when finished.**

Each editor holds a reference to an asset. If you don't detach the editor some asset operations may fail. For example, if you try to delete an asset, but still have an editor hanging on to the asset, the delete function will fail (message will be logged in the K2's log).

C# Code

```
// get a MediaMgr object
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

// get an Editor for a clip
```

```
string clipname = "edl/cmf//local/V:/default/Clip";
ISimpleEditor editor = mediaMgr.CreateEditor(clipname);

// do work

// you must detach your editor when you are done. If you don't you
// will hang onto a reference to the asset and some operations
// will fail (e.g. DeleteAsset).
editor.Detach();

// cleanup
editor.Dispose();
mediaMgr.Dispose();
```

C++ Code

```
// get a MediaMgr object
IMediaMgrPtr spMediaMgr;
hr = spAppServer->CreateMediaMgr(m_bstrAppName, &spMediaMgr);

// get an Editor for a clip
ISimpleEditorPtr spEditor;
hr = spMediaMgr->CreateEditor(_bstr_t("edl/cmf//local/V:/default/Clip"), &spEditor);

// do work

// you must detach your editor when you are done. If you don't you
// will hang onto a reference to the asset and some operations
// will fail (e.g. DeleteAsset).
spEditor->Detach();
```

6.4 Setting Marks

C# Code

```
// set mark in
editor.SetMarkIn(300); // fields offset from zero -OR-
editor.SetMarkIn("00:00:05.00"); // timecode string

// set mark out
editor.SetMarkOut(600); // fields offset from zero -OR-
editor.SetMarkOut("00:00:10.00"); // timecode string
```

C++ Code

```
_variant_t markInFields((long)300); // field offset from zero
_variant_t markInTimecode("00:00:05.00"); // timecode string

_variant_t markOutFields((long)600); // field offset from zero
_variant_t markOutTimecode("00:00:10.00"); // timecode string

HRESULT hr;

// set mark in
hr = spEditor->SetMarkIn(markInFields); // -OR-
hr = spEditor->SetMarkIn(markInTimecode);

// set mark out
hr = spEditor->SetMarkOut(markOutFields); // -OR-
hr = spEditor->SetMarkOut(markOutTimecode);
```

6.5 Creating subclips

C# Code

```
// create subclip
editor.CreateSubclip("//edl/cmf/V:/default/Clip-1", 600, 1200); // fields
editor.CreateSubclip("//edl/cmf/V:/default/Clip-1", "00:00:05.00", "00:00:10.00"); // timecode
```

C++ Code

```
_variant_t markInFields((long)300); // fields
_variant_t markOutFields((long)600); // fields
_variant_t markInTimecode("00:00:05.00"); // timecode string
_variant_t markOutTimecode("00:00:10.00"); // timecode string
_bstr_t bstrSubClip("//edl/cmf/V:/default/Clip-1");

HRESULT hr;

hr = spEditor->CreateSubclip(bstrSubClip, markInFields, markOutFields ); // fields -OR-
hr = spEditor->CreateSubclip(bstrSubClip, markInTimecode, markOutTimecode ); // timecode
```

6.6 Erasing unused media

EraseUnusedMedia is used to delete media not being used by assets. Calling an Editor's EraseUnusedMedia function performs these functions:

1. If the asset is clip and the MarkIn and MarkOut values are set, calling EraseUnusedMedia on the clip's editor will delete the media outside of the marks.

2. If the asset is a subclip, calling EraseUnusedMedia on the subclip's editor will delete all clip media not used by any of the clip's subclips.

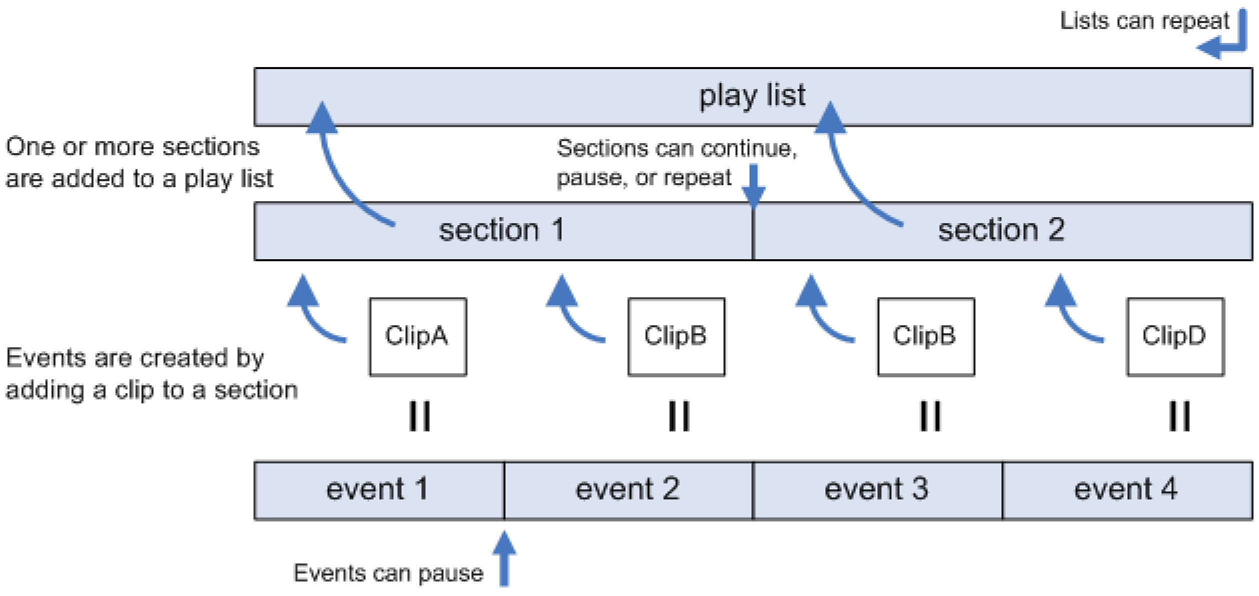
C# Code

```
// erase unused media
editor.EraseUnusedMedia();
```

C++ Code

```
// erase unused media
HRESULT hr = spEditor->EraseUnusedMedia();
```

6.7 Playlists


Lists can repeat
play list
One or more sections are added to a play list
Sections can continue, pause, or repeat
section 1
section 2
ClipA
ClipB
ClipB
ClipD
Events are created by adding a clip to a section
event 1
event 2
event 3
event 4
Events can pause

Playlist:
K2 play channels can also play a sequence of events called a play list. Events are the components that make up a play list.

Section:
All events in a playlist are contained in sections. Playlists must contain at least one section. A playlist can have up to 100 sections. A section can be configured to repeat or pause.

Event:
Events are created by adding a clip or program to sections in a list. Each section can contain up to 1000 events. Events in a playlist can be added, removed or rearranged.

Events can be renamed and trimmed. Trimming an event moves the mark-in and mark-out points. This only affects the event, not the source clip.

A transition between all events in a list is a cut (i.e. the last frame of an event is followed by the first frame of the next event). Events can pause playout at their end. At event pauses, you can choose to show black, freeze on last frame, or freeze on next event.

Program:
Playlists can be saved as a program in the K2 Media Client. Programs created from a playlist include transitions in the playlist, but nothing that breaks the flow of playout, such as a pause at the end of an event. Programs are also created from the loop record mode.

Below is example C# and C++ code for creating a basic playlist. This playlist has two sections that each contain two events.

C# Code
```
// create a two-head player recorder
bool isNewController = false;
ITwoHeadPlayerRecorder player = (ITwoHeadPlayerRecorder) appServer.CreateController(
 "YourApplicationName", "C1", out isNewController);

// set loop play mode to false
player.LoopPlayMode = false;

// enable list pauses
player.SetChannelProperty("listpauses", "true");

// enable list repeats
player.SetChannelProperty("listrepeats", "true");

// set current bin
player.SetCurrentBin("V:", "default");

// generate a unique name for the playlist (i.e. playlist, playlist_0, etc.)
string playlist = player.GenerateUniqueName ("playlist", "_");

// eject the play first to make sure we're clean
player.Eject();

// generate a new clip name (which also loads it)
player.New(playlist);

// get the editor
IEventEditor editor = (IEventEditor) player.GetEditor();

// insert the first section
string section1 = editor.InsertSection("");

// set the starting timecode
editor.SetStartingTimecode("01:00:00,00");

// insert several play events and get properties back
string editID_A = editor.InsertPlayEvent(@"edl/cmf//local/V:/default/ClipA", section1, 0, "");
object[] array = (object[]) editor.GetEventProperty (editID_A, "name+startpos+duration");
// your code to do something with event properties

string editID_B = editor.InsertPlayEvent(@"edl/cmf//local/V:/default/ClipB", section1, 0, "");
array = (object[]) editor.GetEventProperty (editID_B, "name+startpos+duration");
// your code to do something with event properties

// insert another section
string section2 = editor.InsertSection("");

// insert several more play events and get properties back
string editID_C = editor.InsertPlayEvent(@"edl/cmf//local/V:/default/ClipC", section2, 0, "");
array = (object[]) editor.GetEventProperty (editID_C, "name+startpos+duration");
// your code to do something with event properties

string editID_D = editor.InsertPlayEvent(@"edl/cmf//local/V:/default/ClipD", section2, 0, "");
array = (object[]) editor.GetEventProperty (editID_D, "name+startpos+duration");
```

```
// your code to do something with event properties

// play the list for awhile
player.Play();
Thread.Sleep(30000);
player.Stop();

// delete events
editor.DeleteEvent(editID_C);
editor.DeleteEvent(editID_D);

// delete section 2
editor.DeleteSection(section2);

// note: you cannot delete all sections. Since this is a playlist
// it will force you to keep at least one section in it.
// editor.DeleteSection(section2); // this would error

// eject the clip
player.Eject();

// cleanup
editor.Detach();
editor.Dispose();

// close the channel
player.CloseChannel();
```

C++ Code

```
// create a two-head player recorder
short nIsNewController;
ISimpleControllerPtr spController;
_variant_t varChanNum(m_bstrChannel);

// create the controller and cast it to different interfaces
HRESULT hr = spAppServer->CreateController(_bstr_t("YourApplicationName"), _bstr_t("C1"),
 &nIsNewController, &spController);
//printf("CreateController hresult = 0x%x\n", hr);

ISimplePlayerRecorderPtr spPlayer = (ISimplePlayerRecorderPtr) spController;
ITwoHeadPlayerRecorderPtr spTwoHeadPlayer = (ITwoHeadPlayerRecorderPtr) spController;

// set loop play mode to false
VARIANT_BOOL loopmode(FALSE);
hr = spPlayer->put_LoopPlayMode(loopmode);

// enable list pauses
hr = spController->SetChannelProperty(_bstr_t("listpauses"), _variant_t("true"));

// enable list repeats
hr = spController->SetChannelProperty(_bstr_t("listrepeats"), _variant_t("true"));

// set current bin
hr = spController->SetCurrentBin(_bstr_t("V:"), _bstr_t("default"));

// generate a unique name for the playlist (i.e. playlist, playlist_0, etc.)
BSTR playlist;
hr = spController->GenerateUniqueName (_bstr_t("playlist"), _bstr_t("_"), &playlist);

// eject the play first to make sure we're clean
hr = spPlayer->Eject();

// create the new playlist (which also loads it)
hr = spPlayer->New(playlist);

// get an editor and cast it to an event editor
ISimpleEditorPtr spEditor;
hr = spPlayer->GetEditor(&spEditor);
IEventEditorPtr spEventEditor = (IEventEditorPtr) spEditor;

// insert the first section
BSTR section1;
hr = spEventEditor->InsertSection(_bstr_t(""), &section1);
```

```
// set the default starting timecode (this is what AppCenter does).
spEditor->SetStartingTimecode(_variant_t("01:00:00,00"));

// insert several play events and get properties back
BSTR editID_A;
VARIANT varResult;
hr = spEventEditor->InsertPlayEvent(_bstr_t("edl/cmf//local/V:/default/ClipA"), section1, 0, _bstr_t(""),
&editID_A);
hr = spEventEditor->GetEventProperty (editID_A, _bstr_t("name+startpos+duration"), &varResult);
// your code to do something with event properties

BSTR editID_B;
hr = spEventEditor->InsertPlayEvent(_bstr_t("edl/cmf//local/V:/default/ClipB"), section1, 0, _bstr_t(""),
&editID_B);
hr = spEventEditor->GetEventProperty (editID_B, _bstr_t("name+startpos+duration"), &varResult);
// your code to do something with event properties

// insert another section
BSTR section2;
hr = spEventEditor->InsertSection(_bstr_t(""), &section2);

// insert several more play events and get properties back
BSTR editID_C;
hr = spEventEditor->InsertPlayEvent(_bstr_t("edl/cmf//local/V:/default/ClipA"), section1, 0, _bstr_t(""),
&editID_C);
hr = spEventEditor->GetEventProperty (editID_C, _bstr_t("name+startpos+duration"), &varResult);
// your code to do something with event properties

BSTR editID_D;
hr = spEventEditor->InsertPlayEvent(_bstr_t("edl/cmf//local/V:/default/ClipB"), section1, 0, _bstr_t(""),
&editID_D);
hr = spEventEditor->GetEventProperty (editID_D, _bstr_t("name+startpos+duration"), &varResult);
// your code to do something with event properties

// play for awhile
hr = spPlayer->Play();
Sleep(30000);

// eject
hr = spPlayer->Eject();

// cleanup
hr = spEditor->Detach();

// close channel
hr = spController->CloseChannel();
```

6.8 Dual context playing

Each channel has two contexts: a foreground "player" context and a background "preview" context. Contexts can be used schedule transitions in different ways than playlists.

C# Code

```
using Interop.TimecodeLib; // used for timecode object in example 3

// create a controller - cast to player (main context) and preview (background context)
string channel = "C1";
bool isNewController = false;
ISimpleController controller = iappServer.CreateController(
    appName, channel, out isNewController);
ISimplePlayerRecorder player = (ISimplePlayerRecorder) controller;
ITwoHeadPlayerRecorder preview = (ITwoHeadPlayerRecorder) controller;

// load a clip, cue it up, and play
player.Load("edl/cmf//local/V:/default/test1");
player.CueStart();
player.Play();

// while main context is playing, load preview context with 3rd clip
preview.LoadPreview("edl/cmf//local/V:/default/test2");
```

```
// Example 1: tell preview context to "follow" play context
// i.e. preview context starts playing after main context ends
preview.BeginPreviewContextChange();
preview.CueStart(); // cue to some position
preview.Play();
preview.CommitContextChange("follow"); // manual or follow


/*
// Example 2: tell preview context to go active immediately
// then force an immediate context change i.e. asap
preview.BeginPreviewContextChange();
preview.Play();
preview.CommitContextChange("asap");
*/


/*
// Example 3: schedule preview context go active at specific time
string timeOfDay = (string) player.GetStatus("timeofdaystr");
string videoFormat = (string) player.GetStatus("videoformatstr");

// create a new timecode object & set its video format
VdrTimecode timecode = new VdrTimecode();
timecode.VideoFormat = videoFormat;

// get current time, add 5 seconds to it
timecode.AsString = timeOfDay;
timecode.Add("00:00:05,00");

// schedule the context change for 5 seconds from now
preview.BeginPreviewContextChange();
preview.CueStart();
preview.Play();
preview.ScheduleContextChange("timeofday", timecode.AsString);
player.CancelScheduledChange ("timeofday", timecode.AsString);
*/

// wait awhile just to see the transition...
Thread.Sleep(25000);

// eject the clip
player.Stop();
player.Eject();

// close the channel
controller.CloseChannel();
```

C++ Code
TBD

7 Asset management

7.1 File system layout

7.1.1 Volumes, Bins, Assets

See **Error! Reference source not found. Error! Reference source not found.** for an explanation of naming convention. The K2's file system is comprised of a volume called "V:". This volume contains multiple bins that contain assets. The K2's files system only has a bin depth of one level. So bins will always only contain assets, not other bins.

7.1.2 Permanent bins

There are two bins that are always present:

Default:	a default bin that ships with the K2. Unless changed in AppCenter, new assets will be recorded to this bin.
Recycled Bin:	like Windows, this is the bin where assets end up when they are deleted. In AppCenter if you would like to permanently delete an asset, hold the "shift" key down while deleting asset(s).

7.1.3 Enumerating volumes, bins, or assets

To get a list of volumes, bins, or assets you use these MediaMgr calls:

EnumerateVolumes	– gets a list of volumes
EnumerateBins	– gets a list of bins in a volume. Requires volume name
EnumerateAssets	– gets a list of assets in a bin. Requires volume name and bin name.

To use the calls:

1. Call MediaMgr.EnumerateXXX function. This will return a Cookie pointer to the list
2. Call either:
   a. MediaMgr.GetResultProperty with a query string to get back the properties that you want. Results will be returned in an object list, or
   b. MediaMgr.GetXmlResults to get back all properties. Results will be returned in an XML document format.
3. ⚠ After you are finished getting results you <u>MUST</u> close the result set. Each MediaMgr is given a maximum of 10 database selectors. Each time you make EnumerateXXX calls you use a database selector. You release the selector by calling CloseResults on the cookie. If you do not do this you will eventually run out of database selectors and your EnumerateXXX calls will fail!

Below are examples demonstrating how to use GetResultProperty and GetXmlResults.

7.1.3.1 Enumerate assets; get results using GetResultProperty

<u>C# Code</u>

```
// create a MediaMgr object
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

// enumerate the assets in the v:\default bin
int count = 0;
int cookie = mediaMgr.EnumerateAssets("V:", "default", ref count);

// let's query for 3 properties of the clip: the name, video compression type,
// and the length
string query = "name+videoCompressionType+lengthstr";

// Other properties you can query:
// "uri", "name", "assettype", "created", "modified", "attributes1",
// "attributes2", "videoformat", "videocompressiontype", "firsttimecodemask",
// "lengthstr", "maxlength", "markin", "markout", "markinstr", "markoutstr",
// "dropframe", "locked", "hidef"
// for other properties search the guide for "Asset properties"

// Let's only get info on the first 100 clips in the bin
int maxCount = 100;
object dataObj = null;

// Now, let's query for the results we want:
// query it for the properties we setup above,
// start with the 0th element,
// go up to a max of 100 clips
// capture the actual number of clips we're getting data back from,
// and return the results in a data object
mediaMgr.GetResultProperty( cookie, query, 0, maxCount, out count, out dataObj);

Console.WriteLine("Found " + count + " clips in V:\\default\\");

// GetResultProperty for clip data returns a 2 dimensional array
// - the first dimension represents the clip
// - the second dimension represents the properties of the clip

// so let's cast the dataobj to a clip array first
object[] clipArray = (object[]) dataObj;

// loop thru each clip
foreach (object clip in clipArray)
{
 // then cast the clip object to an array of properties for each clip
 object[] propertyArray = (object[]) clip;

 // and finally print out the clip properties for each clip
 Console.WriteLine("Clip: {0} Created: {1} Length: {2}",
 propertyArray[0], propertyArray[1], propertyArray[2]);
}

// make sure that you close the results after enumerating volumes, bins, or assets.
// if you don't, you will run out of database selectors!
mediaMgr.CloseResults(cookie);
mediaMgr.Dispose();
```

C++ Code

```
// create a mediamgr object
_bstr_t bstrAppName("YourAppName");

IMediaMgrPtr spMediaMgr;
HRESULT hr = spAppServer->CreateMediaMgr(bstrAppName, &spMediaMgr);

// enumerate the assets in the v:\default bin
long cookie;
long count = 0;
BSTR bstrXmlData = NULL; // must set this to NULL!

spMediaMgr->EnumerateAssets(_bstr_t("V:"), _bstr_t("default"), &count, &cookie);

// let's query for 3 properties of the clip: the name, video compression type,
// and the length
_bstr_t sQuery("name+videoCompressionType+lengthstr");

// Other properties you can query:
// "uri", "name", "assettype", "created", "modified", "attributes1",
// "attributes2", "videoformat", "videocompressiontype", "firsttimecodemask",
// "lengthstr", "maxlength", "markin", "markout", "markinstr", "markoutstr",
// "dropframe", "locked", "hidef"
// for other properties search the guide for "Asset properties"

// Let's only get info on the first 100 clips in the bin
int maxCount = 100;

// Now, let's query for the results we want:
// query it for the properties we setup above,
// start with the 0th element,
// go up to a max of 100 clips
// capture the actual number of clips we're getting data back from,
// and return the results in a data object
VARIANT varResult;
spMediaMgr->GetResultProperty( cookie, sQuery, 0, maxCount, &count, &varResult);

printf("Found %d clips in V:\\default\\\n", count);

// create a safe array for the clips
SAFEARRAY *pClipArray = varResult.parray;
VARIANT *pVar = NULL;
long arrClipIndex[1];

// get the first clip
arrClipIndex[0] = 0;
SafeArrayPtrOfIndex(pClipArray, arrClipIndex, (void **)&pVar);

// create a safe array for the clip's properties
SAFEARRAY* pPropertyArray = pVar->parray;
long arrPropertyIndex[1];

// get the first property
arrPropertyIndex[0] = 0;
SafeArrayPtrOfIndex(pPropertyArray, arrPropertyIndex, (void **)&pVar);
CString sName = pVar->bstrVal;

// get the second property
arrPropertyIndex[0] = 1;
SafeArrayPtrOfIndex(pPropertyArray, arrPropertyIndex, (void **)&pVar);
CString sVideoCompressionType = pVar->bstrVal;
```

```
// get the third property
arrPropertyIndex[0] = 2;
SafeArrayPtrOfIndex(pPropertyArray, arrPropertyIndex, (void **)&pVar);
CString sLength = pVar->bstrVal;

printf("Clip name: %s video compression type: %s length: %s\n\n", sName, sVideoCompressionType, sLength);

// etc...

// make sure that you close the results after enumerating volumes, bins, or assets.
// if you don't, you will run out of database selectors!
spMediaMgr->CloseResults(cookie);
```

7.1.3.2 Enumerate volumes; get results using GetXmlResults

C# Code

```
// create a MediaMgr object
string appName = "YourAppName";
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

// enumerate the volumes
int count = 0;
int cookie = mediaMgr.EnumerateVolumes(ref count);

// You want to loop thru the results list in blocks no
// bigger than 512 items at a time. The code below shows
// how to do this. This restriction has to do with
// the performance of DOMDocuments degrading after a
// certain size. This becomes important in cases where
// you are getting a list of 10,000 clips from a bin.
// See this link for more details:
// http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BGetXmlResults%20performance%5D%5D

// look thru all XML results. start at result 0 and get
// results in blocks of 512 items. continue thru all
// blocks until we've got them all.

// change the maxCount for different result sizes
int maxCount = 512;    // block size
int i = 0;    // used to indicate where to start
string xmlData = "";
count = maxCount;
while (count == maxCount)
{
    mediaMgr.GetXmlResults(cookie, i * maxCount, maxCount, ref count,
        out xmlData);
    Console.WriteLine(xmlData);
    i++;
}

// very important! close result when done
mediaMgr.CloseResults(cookie);

// cleanup
mediaMgr.Dispose();
```

C++ Code

TBD

7.1.4 Creating and deleting bins

C# Code

```
// create a mediamgr object
string appName = "YourAppName";
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

// create the bin V:/testbin
mediaMgr.CreateBin("V:", "testbin");

// delete the bin V:/testbin
// mediaMgr.DeleteBin("V:", "testbin");

// cleanup
mediaMgr.Dispose();
```

C++ Code

```
// create a mediamgr object
IMediaMgrPtr spMediaMgr;
hr = spAppServer->CreateMediaMgr(_bstr_t("YourApplicationName"),
 &spMediaMgr);

// create the bin V:/testbin
spAppServer->CreateBin( _bstr_t("V:"), _bstr_t("testbin") );

// delete the bin V:/testbin
spAppServer->DeleteBin( _bstr_t("V:"), _bstr_t("testbin") );
```

7.2 Asset management

7.2.1 Renaming, Moving, Copying, and Deleting assets

C# Code

```
// create a MediaMgr object
string appName = "YourAppName";
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

// copy asset
mediaMgr.CopyAsset("edl/cmf//local/V:/default/Clip",
 "edl/cmf//local/V:/default/CopiedClip");

// rename asset
mediaMgr.RenameAsset("edl/cmf//local/V:/default/CopiedClip",
 "RenamedClip");

// move asset
mediaMgr.RenameAsset("edl/cmf//local/V:/default/RenamedClip ", "edl/cmf//local/V:/Recycle Bin/RenamedClip");

// delete asset
mediaMgr.DeleteAsset("edl/cmf//local/V:/Recycle Bin/RenamedClip");

// cleanup
mediaMgr.Dispose();
```

C++ Code

```
// create a MediaMgr object
IMediaMgrPtr spMediaMgr;
HRESULT hr = spAppServer->CreateMediaMgr(_bstr_t("YourAppName"), &spMediaMgr);

// copy asset
hr = spMediaMgr->CopyAsset(_bstr_t("edl/cmf//local/V:/default/Clip"),
  _bstr_t("edl/cmf//local/V:/default/CopiedClip") );

// rename asset
hr = spMediaMgr->RenameAsset(_bstr_t("edl/cmf//local/V:/default/CopiedClip"),
  _bstr_t("RenamedClip") );

// move asset
hr = spMediaMgr->CopyAsset(_bstr_t("edl/cmf//local/V:/default/RenamedClip"),
  _bstr_t("edl/cmf//local/V:/Recycle Bin/RenamedClip") );

// delete asset
hr = spMediaMgr->DeleteAsset(_bstr_t("edl/cmf//local/V:/Recycle Bin/RenamedClip"));
```

7.3 Asset properties

The IMediaMgr GetProperty function can be used to return one or more asset properties. A list of asset properties can be found here:

http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BAsset%20properties%5D%5D

⚠ **Note that if you need to get multiple properties it is more efficient to make one GetProperty call for all properties rather than multiple GetProperty calls for each property.**

C# Code
```
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

// get a single property
string assetType = (string) mediaMgr.GetProperty(
    "edl/cmf//local/V:/default/Clip", "assetType");
Console.WriteLine("asset type = " + assetType);


// get multiple properties
object[] results = (object[]) mediaMgr.GetProperty("edl/cmf//local/V:/default/Clip",
 "LengthStr+videoCompressionType+videoFormatStr");

// loop thru all properties in results list and print them out
foreach (object obj in results)
{
    Console.WriteLine("obj = " + obj);
}

// cleanup
mediaMgr.Dispose();
```

C++ Code
```
IMediaMgrPtr spMediaMgr;
hr = spAppServer->CreateMediaMgr(_bstr_t("YourApplicationName"),
 &spMediaMgr);

VARIANT varResult;
// get a single property
hr = spMediaMgr->GetProperty(_bstr_t("edl/cmf//local/V:/default/Clip"),
 _bstr_t("assetType"), &varResult);

CString sAssetType = varResult.bstrVal;
printf("asset type: %s\n", sAssetType);


// get multiple properties
hr = spMediaMgr->GetProperty(_bstr_t("edl/cmf//local/V:/default/Clip"),
 _bstr_t("LengthStr+videoCompressionType+videoFormatStr"), &varResult);

SAFEARRAY *pArray = varResult.parray;
VARIANT *pVar = NULL;
long arrIndex[1];

// get the first property
arrIndex[0] = 0;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
CString sLength = pVar->bstrVal;

// get the second property
arrIndex[0] = 1;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
CString sVideoCompressionType = pVar->bstrVal;

// get the third property
arrIndex[0] = 2;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
CString sVideoFormat = pVar->bstrVal;

printf("Length: %s VideoCompressionType: %s VideoFormat: %s\n\n",
 sLength, sVideoCompressionType, sVideoFormat);
```

7.4 Getting change notifications

When you get an instance of a MediaMgr, it keeps track of event notifications (i.e. changes in assets, bins, edits, etc.). Calling MediaMgr's GetXmlChanges returns an XML document of all notifications since the last time you called it. The next time GetXmlChanges is called, only new notifications will be returned. Note that each MediaMgr object keeps track of its own list of notifications.

Currently, the best way for getting event notifications is to call GetXmlChanges on a regular interval (say, every half second).

The different event types returned by GetXmlChanges can be found in 0.

For example, let's say you want to keep track of all new clips or changes to clips. Calling GetXmlChanges and querying for notices with EventType="MovieContent" returns this information plus all attributes of the clips. This is an efficient way of finding out what has changed rather than querying individual properties of clips.

Here are some code examples of how to call GetXmlChanges:

C# Code

```
// create a new MediaMgr object
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

// wait 10 seconds - try recording a new asset during this delay
Thread.Sleep(10000);

// get the XML changes and print it out
int changeCount = 0;
string xmlData = "";
mediaMgr.GetXmlChanges(1000000, ref changeCount, out xmlData);
Console.WriteLine(xmlData);

// cleanup
mediaMgr.Dispose();
```

C++ Code

```
// create a new MediaMgr object
IMediaMgrPtr spMediaMgr;
hr = spAppServer->CreateMediaMgr(m_bstrAppName, &spMediaMgr);

// wait 10 seconds - try recording a new asset during this delay
Sleep(10000);

// get the XML changes
long count = 0;
BSTR bstrXmlData = NULL; // must set this to NULL!
spMediaMgr->GetXmlChanges(1000000, &count, &bstrXmlData);

// print it out
CString strXmlData = bstrXmlData;
printf("%s\n", strXmlData);
```

7.5 Writing an efficient clip cache

Many people need to keep track of the contents of a bin. The best way to do this is to write a clip cache in your application. To do this:

First:
- Create a MediaMgr object that you're going to hang onto and use for the life of your application.
- Call MediaMgr.EnumerateAssets to get all clip info from a bin
- Loop thru the clip info results list and build your own cache data structure and store the clip info in your cache

Then:
- Regularly check MediaMgr.GetXmlChanges calls to look for any clip changes
- Update your cache with the change info.

You're cache needs to handle locking to make sure that updating K2 asset information does not interfere with your application getting data from the cache.

8 Transferring, importing, and exporting assets

8.1 Transfer call syntax

The K2 contains a service called the Transfer Queue. Its job is to queue up and execute asset transfer, import, and export requests. The K2 can perform up to 4 simultaneous transfers. The Transfer Queue manages when transfers start. Below is the basic call syntax for adding a transfer request to the Transfer Queue:

```
void IXfrQueue.AddXfrItem(
   string srcUml,
   string destUml,
   string type,
   out int token
);
```

Parameters:

srcUml	the source UML (see 2.8.9 for UML definition)
destUml	the destination UML
type	a string that tells the type of transfer you want to perform
token	returned token for querying the transfer's status

8.2 K2 & Profile transfers

The code below shows how to add a transfer request to transfer an asset between two K2s:

C# Code

```
// create a transfer queue object
IXfrQueue ixfrQueue = iappServer.CreateXfrQueue(appName);

// transfer a clip from between two K2s; get a token back from the
// TransferQueue so that we can query properties of the transfer
int token;
ixfrQueue.AddXfrItem(
"K2-1/explodedFile/V:/default/Clip",
"K2-2/explodedFile/V:/default/Clip",
"Send to Stream",
out token);
```

C++ Code

```
// create a transfer queue object
IXfrQueuePtr spXfrQueue;
HRESULT hr = spAppServer->CreateXfrQueue(m_bstrAppName, &spXfrQueue);

// transfer a clip from between two K2s; get a token back from the
// TransferQueue so that we can query properties of the transfer
long token;
spXfrQueue->AddXfrItem(
 _bstr_t("K2-1/explodedFile/V:/default/Clip"),
 _bstr_t("K2-2/explodedFile/V:/default/Clip"),
 _bstr_t("Send to Stream"),
 &token);
```

8.3 Monitoring transfers

When you add a transfer request you get a token back. The code below shows how you can use this token to query properties of the transfer.

You can get a list of transfer properties that you can query here:
http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BTransfer%20Item%20properties%5D%5D

C# Code

```
// query for a property
object data;
ixfrQueue.GetXfrItemProperty(token, "percentage", out data);
Console.WriteLine("percentage = " + data);
```

```
// get xml data about the transfer item
string xmlData;
ixfrQueue.GetXfrItemXml(token, out xmlData);
Console.WriteLine("result = " + xmlData);
```

C++ Code
```
// query for a property
VARIANT data;
spXfrQueue->GetXfrItemProperty(token, _bstr_t("Percentage"), &data);
long nPercentage = data.lVal;
printf("percentage = %d\n", nPercentage);

// get xml data about the transfer item
BSTR xmlData;
spXfrQueue->GetXfrItemXml(token, &xmlData);

CString sXmlData = xmlData;
printf("xmlData = %s\n\n", sXmlData);
```

8.4 GXF

To export a GXF file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/explodedFile/V:/default/Clip", "K2-1/gxfFile/C:/temp/Clip.gxf",
 "Send to File", out token);
```

To import a GXF file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/gxfFile/C:/temp/Clip.gxf", "K2-1/explodedFile/V:/default/Clip",
 "Import File", out token);
```

8.5 MXF

To export an MXF file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/explodedFile/V:/default/Clip", "K2-1/mxfFile/C:/temp/Clip.mxf",
 "Send to File", out token);
```

To import an MXF file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/mxfFile/C:/temp/Clip.mxf", "K2-1/explodedFile/V:/default/Clip",
 "Import File", out token);
```

8.6 MOV

To export a MOV file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/explodedFile/V:/default/Clip", "K2-1/movFile/C:/temp/Clip.mov",
 "Send to File", out token);
```

To import a MOV file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/movFile/C:/temp/Clip.mov", "K2-1/explodedFile/V:/default/Clip",
 "Import File", out token);
```

8.7 AVI

To export an AVI file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/explodedFile/V:/default/Clip", "K2-1/aviFile/C:/temp/Clip.avi",
 "Send to File", out token);
```

To import an AVI file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/aviFile/C:/temp/Clip.avi", "K2-1/explodedFile/V:/default/Clip",
 "Import File", out token);
```

8.8 MPEG

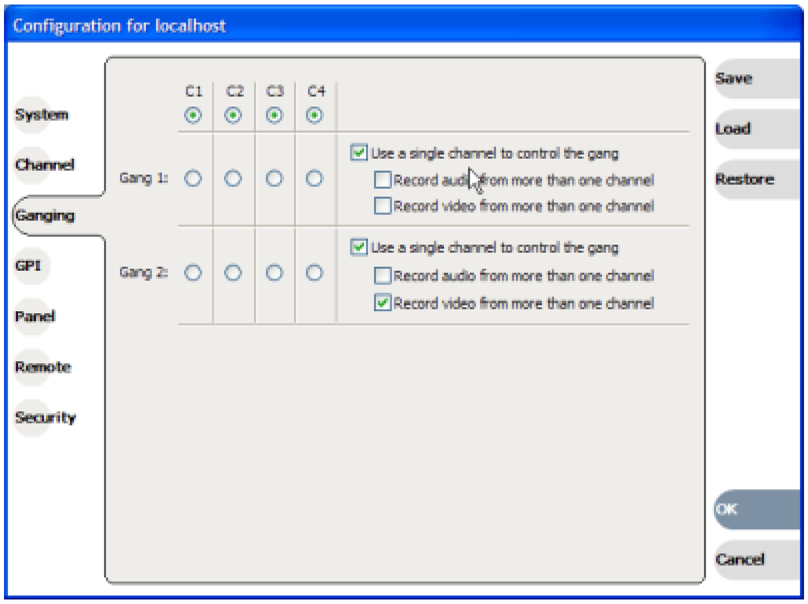
The K2 does not support exporting an MPEG. To import an MPEG file:

C# Code
```
ixfrQueue.AddXfrItem(
 "K2-1/mpgFile/C:/temp/Clip.mpg", "K2-1/explodedFile/V:/default/Clip",
 "Import File", out token);
```

9 Advanced features

9.1 Channel ganging

The image below is the channel ganging window in AppCenter Pro configuration. You will see the Ganging option in the Configuration panel only if you have AppCenter Pro licensed for your K2. Contact GVDeveloperSupport@thomson.net for more information.



	C1	C2	C3	C4
	●	●	●	●
Gang 1:	○	○	○	○
Gang 2:	○	○	○	○

Gang 1:
- [✓] Use a single channel to control the gang
- [ ] Record audio from more than one channel
- [ ] Record video from more than one channel

Gang 2:
- [✓] Use a single channel to control the gang
- [ ] Record audio from more than one channel
- [✓] Record video from more than one channel

To get or set channel ganging properties via the AppServer API do the following:

If you do not own the channel, your calls are limited to querying whether a channel is ganged and to which gang it belongs from the channel status interface:

C# Code
```
// create a chanstatus object
string appName = "YourApplicationName";
string channel = "C1";
IChanStatus ichanStatus = iappServer.CreateChanStatus(appName, channel);

// get multiple properties
object[] results = (object[]) ichanStatus.GetStatus("ganged+gangindex");

// "ganged" is either true or false
// when not ganged, the "gangindex" == -1
// belonging to gang 1, the "gangindex" == 0
// belonging to gang 2, the "gangindex" == 1
foreach (object obj in results)
{
  Console.WriteLine("obj = " + obj);
}

// cleanup
ichanStatus.Dispose();
```

C++ Code

```
// create a chanstatus object
IChanStatusPtr spChanStatus;
HRESULT hr = spAppServer->CreateChanStatus(_bstr_t("YourApplicationName"),
 _bstr_t("C1"), &spChanStatus);

// get multiple properties
VARIANT varResult;
hr = spChanStatus->GetStatus(_bstr_t("ganged+gangindex"), &varResult);

// "ganged" is either true or false
// when not ganged, the "gangindex" == -1
// belonging to gang 1, the "gangindex" == 0
// belonging to gang 2, the "gangindex" == 1
SAFEARRAY *pArray = varResult.parray;
VARIANT *pVar = NULL;
long arrIndex[1];

arrIndex[0] = 0;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
BOOL bGanged = (pVar->boolVal == VARIANT_TRUE);

arrIndex[0] = 1;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
int nGangIndex= pVar->iVal;

printf("Ganged: %d Gang Index: %d\n\n", bGanged, nGangIndex);
```

If you want to set the gang properties, you'll need to create a controller for the channel and set or get properties like this:

C# Code

```
// create a controller
string appName = "YourApplicationName";
string channel = "C1";
bool isNewController = false;
ISimpleController icontroller = iappServer.CreateController(
 appName, channel, out isNewController);

// not ganged: set "gangindex" = -1
// gang 1: set "gangindex" = 0
// gang 2: set "gangindex" = 1
icontroller.SetChannelProperty("gangindex", 0);

// to use a single channel to control the gang: set "singlecontroller" = true
icontroller.SetChannelProperty("singlecontroller", true);

// to record audio from more than one channel: set "consolidateaudio" = true
icontroller.SetChannelProperty("consolidateaudio", false);

// to record video from more than one channel: set "consolidatevideo" = true
icontroller.SetChannelProperty("consolidatevideo", true);

// get all gang properties of the channel
// note: currently the "singlecontroller" property only returns
// a valid value if the channel IS ganged.
```

```
string query = "ganged+gangindex+singlecontroller+consolidatevideo+consolidateaudio";
object[] results = (object[]) icontroller.GetChannelProperty(query);

// print all returned property values
foreach (object obj in results)
{
  Console.WriteLine("obj = " + obj);
}

// cleanup
icontroller.CloseChannel();
```

C++ Code

```
// create a controller
short nIsNewController;
ISimpleControllerPtr spController;
HRESULT hr = spAppServer->CreateController(_bstr_t("YourApplicationName"),
 _bstr_t("C1"), &nIsNewController, &spController);

// not ganged: set "gangindex" = -1
// gang 1: set "gangindex" = 0
// gang 2: set "gangindex" = 1
spController->SetChannelProperty(_bstr_t("gangindex"), 0);

// tp use a single channel to control the gang: set "singlecontroller" = true
VARIANT_BOOL singlecontroller(FALSE);
spController->SetChannelProperty(_bstr_t("singlecontroller"), singlecontroller);

// to record audio from more than one channel: set "consolidateaudio" = true
VARIANT_BOOL consolidateaudio(FALSE);
spController->SetChannelProperty(_bstr_t("consolidateaudio"), consolidateaudio);

// to record video from more than one channel: set "consolidatevideo" = true
VARIANT_BOOL consolidatevideo(FALSE);
spController->SetChannelProperty(_bstr_t("consolidatevideo"), consolidatevideo);

VARIANT varResult;
// get all gang properties of the channel
// note: currently the "singlecontroller" property only returns
// a valid value if the channel IS ganged.
hr = spController-
>GetChannelProperty(_bstr_t("ganged+gangindex+singlecontroller+consolidatevideo+consolidateaudio"),
 &varResult);

SAFEARRAY *pArray = varResult.parray;
VARIANT *pVar = NULL;
long arrIndex[1];

arrIndex[0] = 0;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
BOOL bGanged = (pVar->boolVal == VARIANT_TRUE);

arrIndex[0] = 1;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
int nGangIndex= pVar->iVal;

arrIndex[0] = 2;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
BOOL bSingleController = (pVar->boolVal == VARIANT_TRUE);

arrIndex[0] = 3;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
BOOL bConsolidateVideo = (pVar->boolVal == VARIANT_TRUE);

arrIndex[0] = 4;
SafeArrayPtrOfIndex(pArray, arrIndex, (void **)&pVar);
BOOL bConsolidateAudio = (pVar->boolVal == VARIANT_TRUE);

printf("Ganged: %d Gang Index: %d Single Controller: %d Consolidate Video: %d Consolidate Audio: %d\n\n",
 bGanged, nGangIndex, bSingleController, bConsolidateVideo, bConsolidateAudio);

// cleanup
hr = spController->CloseChannel();
```

9.2 Audio mapping

The functionality for audio mapping has slightly changed recently. Updated example code will released soon.

C# Code
TBD

C++ Code
TBD

10 Other

10.1 K2 Tips

10.1.1 Keep objects and reuse them

If you have an application where you need to use an object on a regular basis, create the object once and hang onto it. Don't repeatedly create objects and then throw them away.

Example:
Let's say that you have a routine that is regularly called. This routine checks for clips in a bin. The routine runs every 30 seconds. In this routine you use a MediaMgr object to make CheckAsset calls. Rather than creating a new MediaMgr object each time the routine is called, create the MediaMgr object once and store it somewhere in your program.

This will eliminate excessive setup and teardown of objects both in your client application and the K2.

10.1.2 Get multiple properties with one call

For GetProperty or GetStatus calls, multiple properties can be requested by putting more than one string in the property query. This is more efficient than making multiple single property queries.

Property names should be separated by white space or the '+' character. When a single parameter is requested, an object (for C#) or variant (for C++) is returned of the appropriate type. When returning multiple values, an object array (for C#) or a variant of variants (for C++) is returned where each contained object or variant is of the appropriate type. The order of elements in this array matches the order in the property query.

See 4.4.2 Channel properties and 7.3 Asset properties for examples.

10.1.3 Call CloseResults after MediaMgr.EnumerateXXX calls

For the MediaMgr calls EnumerateVolumes, EnumerateBins, and EnumerateAssets, you need to call CloseResults on the cookie when you are finished using it. Your MediaMgr object only gets ten database selectors to work with. Every time you make an EnumerateXXX call you use one of the selectors. Once you run out of selectors, your enumerate calls will fail. You will also see "Too many open select statements: limit is 10" errors in the K2's log. Calling CloseResults frees the selector and makes it usable to MediaMgr again.

See 7.1.3 Enumerating volumes, bins, or assets for an example.

10.1.4 Make efficient calls

Consider what the call does and the frequency that you really need to call it.

Example:
Take the MediaMgr call GetXmlChanges. Its job is to return information on what has changed in the file system since the last time that you called it (i.e. new clips recorded, assets deleted, a clip's marks changed, etc). If you have a thread that calls it every 5 ms there is a good chance that nothing has changed since you last called it and 95% of your GetXmlChanges calls will return empty.

Don't waste your app's and the K2's time by making unnecessarily frequent calls. Consider the rate you really need the information. Remember that every call has to go thru marshall & unmarshall steps which contribute to the system's overall CPU usage.

10.1.5 Audit your application's calls

Use function trace logging to review the calls that you are making to the K2:

- Do you have any duplicate calls that can be eliminated (e.g. can 10 GetProperty calls be replace by 1 with multiple properties)?
- Is the frequency okay (e.g. 100 GetStatus calls a second is bad)?
- Can you refactor any calls (e.g. can you replace 4 different threads all making the same call with just one)?

See 10.5 Enabling function trace logging for more information.

10.2 K2 API

The K2 AppServer API is documented at the link below. This includes namespaces, classes, methods, and properties:

http://www.gvgdevelopers.com/K2DevGuide/API/index.html

10.3 Logging messages

Below is sample code showing how to write debug, status, warning, error, and fatal messages to the K2's log.

C# Code
```
// create a log object
IAppServerLog log = iappServer.CreateLog(appName);

// write different message types
log.LogMsg(EntryPtMsgType.eDebugMsg, 0, "debug message");
log.LogMsg(EntryPtMsgType.eStatusMsg, 0, "status message");
log.LogMsg(EntryPtMsgType.eWarningMsg, 0, "warning message");
log.LogMsg(EntryPtMsgType.eErrorMsg, 0, "error message");
log.LogMsg(EntryPtMsgType.eFatalMsg, 0, "fatal message");

// cleanup
log.Dispose();
```

C++ Code
```
// create a log object
IAppServerLogPtr spLog;
hr = spAppServer->CreateLog(m_bstrAppName, &spLog);

// write different message types
hr = spLog->LogMsg(EntryPtMsgType_eDebugMsg, 0, _bstr_t("debug message"));
hr = spLog->LogMsg(EntryPtMsgType_eStatusMsg, 0, _bstr_t("status message"));
hr = spLog->LogMsg(EntryPtMsgType_eWarningMsg, 0, _bstr_t("warning message"));
hr = spLog->LogMsg(EntryPtMsgType_eErrorMsg, 0, _bstr_t("error message"));
hr = spLog->LogMsg(EntryPtMsgType_eFatalMsg, 0, _bstr_t("fatal message"));
```

10.4 View movie

To view a movie's full structure, you can get an asset's "summary" property. This will return a string representation of detailed information about all media, edits, segments, marks, etc. You can also get an XML representation by querying for the property "xml" instead. See 7.3 Asset properties about querying other asset properties.

C# Code
```
// get a MediaMgr object
IMediaMgr mediaMgr = iappServer.CreateMediaMgr(appName);

string clipname = "edl/cmf//local/V:/default/Clip";

// get V:/default/Clip's summary
string ViewMovieSummary = (string) mediaMgr.GetProperty(clipname, "summary");
Console.WriteLine(ViewMovieSummary);

// cleanup
```

```
mediaMgr.Dispose();
```

C++ Code

```
// get a MediaMgr object
IMediaMgrPtr spMediaMgr;
hr = spAppServer->CreateMediaMgr(_bstr_t("YourApplicationName"),
 &spMediaMgr);

// get V:/default/Clip's summary
VARIANT varResult;
hr = spMediaMgr->GetProperty(_bstr_t("edl/cmf//local/V:/default/Clip"),
 _bstr_t("summary"), &varResult);

CString sSummary = varResult.bstrVal;
printf("Summary: %s\n", sSummary);
```

10.5 Enabling function trace logging

Starting in K2 version 3.2 software any client using the AppServer API can enable function trace logging. This lets you capture a time-stamped log of a client's AppService calls, the parameters being sent, and the data returned from the call.

Uses:
For starters, you can enable function trace logging and then use AppCenter to perform a task. You can then review the function trace log and see what calls and parameters were used to perform each operation. This is a great tool to help learn the calls and parameters of the AppServer API.

Later, in your development cycle, you can enable function trace logging for your application. This helps you to look at your application like a black box and see the calls and the timing of calls that you are making. Especially in a multi-threaded application, this is helpful for verifying:

- If calls are being made to frequently?
- If redundant calls are being made?
- If parameters are wrong or missing?

⚠ **Make sure that function trace logging is disabled in live environments**

Function trace logging is a good tool for evaluating performance, but you should disable it in a live environment. The additional logging calls add to file I/O overhead and could affect performance.

On the K2 Client or Control Point, in the c:\profile directory, you will see three NLog.* files. The AppService proxy objects use NLog to log all calls and parameters as well as the returned results. NLog offers great flexibility to turn on-or-off what gets logged to where based on logging targets and rules. This is configured in the NLog.config file. You can learn more about NLog and it's configuration at http://www.nlog-project.org/.

For example, starting up AppCenter with NLog logging enabled produces a log that looks like this:

```
08-31-2007 15:34:26.865 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy..ctor
08-31-2007 15:34:27.256 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.SetHost ("localhost")
08-31-2007 15:34:27.490 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.SetUserCredentials
("Username", "PASSWORD", "", False)
08-31-2007 15:34:27.552 [Workstation:3852]
GrassValley.Mseries.AppServer.AppServerMgrProxy.GetEncryptedUserCredentials
08-31-2007 15:34:27.662 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.Connect
08-31-2007 15:34:28.459 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.Connect returning:
true
08-31-2007 15:34:28.459 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.IsUserAuthenticated
08-31-2007 15:34:28.568 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.IsUserAuthenticated
returning: True
08-31-2007 15:34:28.568 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.IsUserInRole
("BUILTIN\Administrators")
08-31-2007 15:34:28.662 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.IsUserInRole
returning: True
08-31-2007 15:34:28.677 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.GetSystemStatus
```

```
08-31-2007 15:34:28.677 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.GetSystemStatus
returning: ("Normal", UP_AND_RUNNING, NORMAL)
08-31-2007 15:34:30.099 [Workstation:3852] GrassValley.Mseries.AppServer.AppServerMgrProxy.CreateAppServer ("K2-
1_localConnection", "Workstation")
etc.
```

As you can see, information about every process, process id, class, function, parameters, and results gets logged. This way it is very easy to see the exact commands and parameters to use to perform the operations you want.

To enable this function trace logging:

1. In Explorer go to c:\profile and look for the "NLog.config" file. This is an XML file which specifies the logging target and the rules for logging.

2. Right-click on it and disable the "read only" checkbox.

3. Open NLog.config in a text editor. You'll see that all targets and rules are commented out with XML comments. Good choices to uncomment are the first target with the name "dailylogfile" and the last logger in the rules section which writes most messages, but avoidAppCenter update chatter. Note that the first target writes log entries to the file "C:\logs\AppService_calls.log". This file rotates daily, keeping 2 backup files.

4. Start your application perform the functionality that want, then view all calls in the C:\logs\AppService_calls.log file.

Additionally, you can download a pre-configured NLog.config file from here:
http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BHow%20to%20turn%20on%20function%20logging%20to%20see%20the%20calls%20AppCenter%20is%20making%5D%5D

Note:
The function trace log captures AppServer API calls only. Other protocol calls and mouse and keyboard commands are not captured here.

10.6 ControlPoint PC

Requirements	Comments
Operating System	Microsoft Windows XP Professional, Service Pack 2 (Must be a U.S. version)
RAM	Minimum 512 MB, 1 GB recommended
Graphics acceleration	Must have at least 128M memoryProcessorPentium 4 or higher class, 2 GHz or greater
Hard disk space	400 MB
MS .NET	Version 1.1 and 2.0
Java	JRE1.3.1_12 and 1.4.2_05 or higher. Required for HP Ethernet Switch Configuration interface, which is used for K2 Storage Systems (external storage).
XML	Microsoft XML 4 Service Pack 2

10.7 Scripting with C#

There is an open source program called CS-Script (pronounced C Sharp script) available here:
http://www.members.optusnet.com.au/~olegshilo/

CS-Script is a Common Language Runtime based scripting system which uses real C# code. It combines the power of the C# language with the flexibility of a scripting system.

The advantage is that you can take a single C# file and run it from a command-line like this:

```
    C:\> cscs.exe PlayClip.cs
```

This is a useful tool for scripting tools and test programs for the K2. More information and many sample K2 scripts can be found here:

http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BAsset%20properties%5D%5D

Appendix A: Asset Properties

See Notes at bottom of table

Property	Type	Description	Notes
assetType	BSTR	String indicates the asset type. Values include Clip, Program, and List	
bin	BSTR	Name of the bin that contains the asset.	e
construction	BSTR	Indicates the asset's construction status. Values include: CodecConstruction, CopyConstruction, StreamConstruction, RestoreConstruction, and DoneConstruction.	
created	DATE	Date created.	
dropFrame	VARIANT_BOOL	VARIANT_TRUE if the asset's timecode media indicates drop-frame.	
eventCount	long	Number of events in a List asset.	e
fileLength	long	Difference between maxMarkOut and minMarkIn properties.	e
fileLengthStr	BSTR	Timecode string representation of the difference between maxMarkOut and minMarkIn properties.	e
firstTimecodeMask	long	Cached timecode value from the start of the first timecode media file. Timecode is represented as a 32 bit mask.	
isinuse	VARIANT_BOOL	VARIANT_TRUE if the movie is locked, has another writable instance open, is under construction, or is loaded on the system processing the property query.	
lastTimecodeMask	long	Cached timecode value from the end of the last timecode media file. Timecode is represented as a 32 bit mask.	
lengthStr	BSTR	String representing the MaxLength value converted to timecode. A ';' character before the frame value indicates dropframe.	
locked	VARIANT_BOOL	VARIANT_TRUE if the asset is locked.	
markIn	long	Asset mark in value. Does not necessarily reflect media marks.	
markInStr	BSTR	String representation of the timecode from the start of the first timecode media file on the first timecode track. A ';' or ',' character before the frame value indicates dropframe.	
markOut	long	Asset mark out value. Does not necessarily reflect media marks.	
markOutStr	BSTR	String representation of the timecode from the end of the last timecode media file on the first timecode track. Note that this value is one frame greater than the last playable timecode frame. A ';' or ',' character before the frame value indicates dropframe.	
maxLength	long	Number of fields in longest track (the sum of all media durations).	
maxMarkOut	long	This is the largest value the mark out can be set to and still show media on all tracks.	e
maxMarkOutStr	BSTR	This is a string representation of the smallest timecode value the mark in can be set to and still show media on all tracks. The timecode value is mathematically calculated using the firstTimecode property.	e
minMarkIn	long	This is the smallest value the mark in can be set to and still show media on all tracks.	e

minMarkInStr	BSTR	This is a string representation of the smallest timecode value the mark in can be set to and still show media on all tracks. The timecode value is mathematically calculated using the firstTimecode property.	e
modified	DATE	Date last modified.	
movieId	BSTR	32 character unique identifier (UUID)	
movieSize	long	The sum of media file sizes and user data in megabytes (MB). If the movie only references a portion of a file, then only a percent of the file size value is used.	e, x
name	BSTR	Asset name	
numA	long	Number of audio tracks.	
numT	long	Number of timecode tracks.	
numV	long	Number of video tracks.	
readOnly	VARIANT_BOOL	VARIANT_TRUE if the asset is locked or is a read-only instance (which can happen if some other application has the asset already open for writing). Applications can only lock and unlock assets. ReadOnly is merely reflects the current state of the asset.	
repeat	VARIANT_BOOL	VARIANT_TRUE if this is a List asset type with the “repeat” property.	
sectionCount	long	Number of sections in a List asset.	e
starttimecode	long	Arbitrary starting timecode for a List asset. Timecode is represented as a 32 bit mask.	e
starttimecodestr	BSTR	Arbitrary starting timecode for a List asset. Blank if a starting timecode was never defined.	e
summary	BSTR	English-based summary of the asset structure. This string is used by utility programs like ViewMovie.	e, x
thumbnail	BYTE SAFEARRAY	Jpeg thumbnail.	e
thumbnailMark	long	Thumbnail mark value. This value is relative to the asset's mark in value. For example, if the thumbnail is taken 32 fields into a clip whose mark in value is 100, then the thumbnail mark value would be 132.	e
underConstruction	VARIANT_BOOL	VARIANT_TRUE if the asset is under any type of construction.	
uri	BSTR	Asset identifier in URI format. The URI string takes the following form: edl/cmf//<machine>/<volume>/<bin>/<name>	e
version	long	Version of this property list. Currently set to 1.	
videoCompressionType	BSTR	Compression type of the first video track. Values include Mpeg, DV25, DV50, Jpeg, and MpegHiDef.	
videoFormat	long	Asset video format. Enumerated values include: Format525_60Hz_2To1 = 0, Format625_50Hz_2To1, Format720_59_94Hz_1To1, Format720_60Hz_1To1, Format1080_23_98Hz_1To1, Format1080_24Hz_1To1, Format1080_25Hz_1To1, Format1080_29_97Hz_1To1, Format1080_30Hz_1To1, Format1080_25Hz_2To1, Format1080_29_97Hz_2To1, Format1080_30Hz_2To1, Format1035_30Hz_2To1, Format1035_29_97Hz_2To1, VideoFormat_NA, Unknown.	
videoFormatStr	BSTR	Asset video format. Values include: Format525_60Hz_2To1, Format625_50Hz_2To1, Format720_59_94Hz_1To1, Format720_60Hz_1To1, Format1080_23_98Hz_1To1, Format1080_24Hz_1To1, Format1080_25Hz_1To1, Format1080_29_97Hz_1To1, Format1080_30Hz_1To1, Format1080_25Hz_2To1, Format1080_29_97Hz_2To1, Format1080_30Hz_2To1, Format1035_30Hz_2To1, Format1035_29_97Hz_2To1, VideoFormat_NA, Unknown.	
volume	BSTR	Name of the disk volume associated with the asset's bin.	e

xml	BSTR	XML representation of the asset.	e, x

e – Best performance with one of the Editor interfaces (ISimpleEditor, IEventEditor). Properties with this note can still be obtained by IMediaMgr::GetProperty, but extra database transactions have to be made that makes these properties more expensive.

x – Expensive. This properties should only be obtained occasionally and as a result of a user request, rather than on a timer.

Appendix B: Channel Properties

Application [supplied] Status			
Property	**Type**	**Description**	**Notes**
text0	BSTR	Custom text string generally provided by the application that owns the channel	
text1	BSTR	Custom text string generally provided by the application that owns the channel	
text2	BSTR	Custom text string generally provided by the application that owns the channel	
text3	BSTR	Custom text string generally provided by the application that owns the channel	
statusTemplate	BSTR	Status templates string last set with a call to IChanStatus::put_StatusTemplate	
thumbnailStatus	BSTR	Indicates the status of the thumbnail provided by the application that owns the channel. Values include: Empty, New, Same, and Unknown. The status is initialized to "Unknown". It is "New" just after a call to IChanStatus::SetThumbnail. It is "Same" after the first time the thumbnail status is returned by this process. It is "Empty" after a call to IChanStatus::ClearThumbnail.	
thumbnail	BYTE SAFEARRAY	Jpeg thumbnail.	
totalstorage	long	total disk space in MBytes	
remainingstorage	long	remaining disk space in MBytes	
pcntstorageremaining	long	percentage of disk space remaining	
timeremainingstr	BSTR	time remaining on disk displayed as timecode value (depends on how channel is configured)	
timeremaining	long	time remaining in fields (depends on how channel is configured)	

Channel Status			
Property	**Type**	**Description**	**Notes**
channelName	BSTR	Name of this channel (K2: C1, C2, C3, C4; MSeries: R1, P1, R2, P2)	
channelType	BSTR	Values include Player, Recorder, and PlayerRecorder.	
ownerName	BSTR	Name of the application that has allocated this channel	
videoFormat	long	Channel's configured video format. Enumerated values include: Format525_60Hz_2To1 = 0, Format625_50Hz_2To1, Format720_59_94Hz_1To1, Format720_60Hz_1To1, Format1080_23_98Hz_1To1, Format1080_24Hz_1To1, Format1080_25Hz_1To1, Format1080_29_97Hz_1To1, Format1080_30Hz_1To1, Format1080_25Hz_2To1, Format1080_29_97Hz_2To1, Format1080_30Hz_2To1, Format1035_30Hz_2To1, Format1035_29_97Hz_2To1, VideoFormat_NA, Unknown.	

videoFormatStr	BSTR	String representation of channel's configured video format. Values include: Format525_60Hz_2To1, Format625_50Hz_2To1, Format720_59_94Hz_1To1, Format720_60Hz_1To1, Format1080_23_98Hz_1To1, Format1080_24Hz_1To1, Format1080_25Hz_1To1, Format1080_29_97Hz_1To1, Format1080_30Hz_1To1, Format1080_25Hz_2To1, Format1080_29_97Hz_2To1, Format1080_30Hz_2To1, Format1035_30Hz_2To1, Format1035_29_97Hz_2To1, VideoFormat_NA, Unknown.	
fieldsPerFrame	long	For interlaced video formats this value is 2. For progressive scan, the value is 1.	
ganged	BOOL	whether or not the channel is ganged	
gangindex	long	-1 = unganged, 0 = gang 1, 1 = gang 2	
singlecontroller	BOOL	whether or not a single channel controls the gang	
consolidateaudio	BOOL	whether or not to record audio from more than one channel	
consolidatevideo	BOOL	whether or not to record video from more than one channel	

Asset Status

Property	Type	Description	Notes
assetname	BSTR	Asset name.	
assetUri	BSTR	Asset identifier in URI format. The URI string takes the following form: edl/cmf//<machine>/<volume>/<bin>/<name>	
editName	BSTR	When the channel is playing a List asset, this is the name of the edit currently being played.	
nextEditName	BSTR	When the channel is playing a List asset, this is the name of the next edit to be being played. This status value is currently always blank.	
sectionName	BSTR	When the channel is playing a List asset, this is the name of the section currently being played.	
nextSectionName	BSTR	When the channel is playing a List asset, this is the name of the next section to be being played. This status value is currently always blank.	

Timeline Status

Property	Type	Description	Notes
durationStr	BSTR	Timecode representation of the channel timeline's maxPos – minPos value. This is the amount of playable material on the timeline.	r
countdownStr	BSTR	Timecode representation of the maxPos – position value. This is the amount of play time left on the timeline.	r
percentComplete	float	Channel timeline's maxPos – position expressed as a percent. If the timeline is being used for a crash record, then the percent complete gets to 100.0 every 10 seconds and starts over at 0.0.	r
position	long	Timeline position in fields. This value will always be 0 or greater.	r
portState	long	Enumerated value that indicates if the timeline is idle, cued for playback, cued for record, playing, or recording.	r
shuttleSpeed	float	Current rate. Normal play speed is 1.0.	r

sectionCountdownStr	BSTR	Timecode representation of the end-of-the-current-section – position value. This is the amount of play time left in the current section.	r
eventCountdownStr	BSTR	Timecode representation of the end-of-the-current-edit – position value. This is the amount of play time left in the current section.	r
sectionPercent	float	Channel timeline's end-of-the-current-section – position value expressed as a percent.	r

Video Status			
Property	**Type**	**Description**	**Notes**
fieldsize	long	Compressed size of the last played field in bytes.	r
pixCoding	BSTR	I, B, or P	r

Audio Status (track 0)			
Property	**Type**	**Description**	**Notes**
inputGain0	float	Audio input gain value. This is used to adjust the input audio level. Units are in dBU. Range is –40.0 to 20.0.	r
outputGain0	float	Audio output gain value. This is used to adjust the output audio level. Units are in dBU. Range is –40.0 to 20.0.	r
audioLevel0	float	Current audio level after gain value has been applied. Units are in dBU.	r

Audio Status (track 1)			
Property	**Type**	**Description**	**Notes**
inputGain1	float	Audio input gain value. This is used to adjust the input audio level. Units are in dBU. Range is –40.0 to 20.0.	r
outputGain1	float	Audio output gain value. This is used to adjust the output audio level. Units are in dBU. Range is –40.0 to 20.0.	r
audioLevel1	float	Current audio level after gain value has been applied. Units are in dBU.	r

Timecode Status			
Property	**Type**	**Description**	**Notes**
timecodeStr	BSTR	When EE or recording, TimecodeStr is the status value from the first timecode resource's current timecode source (VITC, LTC, or INT). When playing back, TimecodeStr represents the timecode media value.	r

Notes:
r – Status is available in a range of values (IChanStatus::GetStatusRange)

Appendix C: Transfer Item properties

The following are properties of a transfer. You can retrieve these properties by calling the TransferQueue's GetXfrItemProperty() function.

Property	**Value**
"state"	0 = pending, 1 = transferring, 2 = completed, 3 = error, 4 = aborted
"percentage"	transfer percentage complete
"averagerate"	average transfer rate
"currentrate"	current transfer rate
"bytestransferred"	bytes transferred so far
"fieldstransferred"	fields transferred so far
"totalbytes"	total bytes
"totalfields"	total fields
"errorcode"	error code if transfer fails – see "state" above.

Appendix D: Event Notifications

The following XML document defines the event notifications you can receive from the K2. See section 7.4 Getting change notifications for an explanation and example to how get the notices.

```
<XmlChanges>
<ns:Data>

<!-- BIN EVENTS -->
<!-- when a bin is created (e.g. create V:\test bin) -->
<ns:Notice EventType="GroupAdded" MachineName="K2-1" DatasetName="V:" GroupName="test" MovieName="~tmp"
ModifyMask="64" EventTime="2007-05-01T06:34:58.000"/>

<!-- when a bin is deleted (e.g. delete V:\test bin) -->
<ns:Notice EventType="GroupDeleted" MachineName="K2-1" DatasetName="V:" GroupName="test" MovieName="~tmp"
ModifyMask="128" EventTime="2007-05-01T06:37:19.000"/>


<!-- ASSET EVENTS -->
<!-- when a clip is created (e.g. create new clip V:\default\Clip) -->
<ns:Notice EventType="MovieAdded" MachineName="K2-1" DatasetName="V:" GroupName="default" MovieName="Clip"
ModifyMask="32768" EventTime="2007-05-01T06:19:42.000"/>

<!-- when a movie is deleted (e.g. delete clip V:\default\Clip)-->
<ns:Notice EventType="MovieDeleted" MachineName="K2-1" DatasetName="V:" GroupName="default" MovieName="Clip"
ModifyMask="65536" EventTime="2007-05-01T06:41:41.000"/>

<!-- when a movie renamed or moved (e.g. clip V:\default\Clip renamed to V:\default\Clip_1) -->
<ns:Notice EventType="MovieRenamed" MachineName="K2-1" DatasetName="V:" GroupName="default" MovieName="Clip"
ModifyMask="16" EventTime="2007-05-01T06:33:18.000">
<ns:RenameNotice GroupName="default" MovieName="Clip_1"/>
</ns:Notice>

<!-- attributes of a clip (e.g. attributes of clip V:\default\Clip) -->
<ns:Notice EventType="MovieContent" MachineName="K2-1" DatasetName="V:" GroupName="default" MovieName="Clip"
ModifyMask="16777216" EventTime="2007-05-01T06:19:42.000">
<ns:MovieContentNotice AssetType="Clip" VideoFormat="Format525_60Hz_2To1" VideoCompressionType="Unknown"
AudioFormat="PCM" AspectRatio="Unknown" AspectRatioConversion="Default" Construction="DoneConstruction"
MovieId="e08ac55c67cd4ace9e5958a83bdf8220" MarkInStr="" MarkOutStr="" LengthStr="00:00:00,00" StartTimeStr=""
MarkIn="0" MarkOut="0" MinMarkIn="0" MaxMarkOut="0" ThumbnailMark="-1" MaxLength="0" AttributeMask1="720896"
AttributeMask2="0" NumVideo="0" NumAudio="0" NumTimecode="0" NumAncData="0" Created="2007-05-01T06:19:41.000"
Modified="2007-05-01T06:19:41.000" Locked="0" DropFrame="0" Repeat="0" UnderConstruction="0"
FirstTimecodeMask="-2147483648"/>
</ns:Notice>

<!-- when a thumbnail of a clip is created (e.g. thumbnail of clip V:\default\Clip) -->
<ns:Notice EventType="MovieThumbnailImage" MachineName="K2-1" DatasetName="V:" GroupName="default"
MovieName="Clip" ModifyMask="131072" EventTime="2007-05-01T06:19:45.000">
<ns:MovieThumbnailNotice thumbnailMark="32"/>
</ns:Notice>

<!-- EXTENSION EVENTS -->
<!-- when an extension is created (e.g. add user data) -->
<ns:Notice EventType="ExtensionAdded" MachineName="K2-1" DatasetName="V:" GroupName="default" MovieName="Clip"
ModifyMask="4096" EventTime="2007-05-01T10:54:55.000">
<ns:ExtensionAddedNotice extensionPosition="-1" extensionCreator="Thomson" extensionClass="Metadata"
extensionName="test" extensionLocation="string" extensionVersion="version 1.2.3"
extensionUuid="7602985a40694837ba0e8b1d330d3bc1"/>
</ns:Notice>

<!-- when an extension is deleted (e.g. remove user data) -->
<ns:Notice EventType="ExtensionDeleted" MachineName="K2-1" DatasetName="V:" GroupName="default" MovieName="Clip"
ModifyMask="4096" EventTime="2007-05-01T10:57:12.000">
<ns:ExtensionDeletedNotice extensionUuid="7602985a40694837ba0e8b1d330d3bc1"/>
</ns:Notice>

<!-- PLAYLIST SECTION EVENTS -->
<!-- when a section is added -->
<ns:Notice EventType="SectionAdded" MachineName="K2-1" DatasetName="V:" GroupName="default"
MovieName="~scratchclip" ModifyMask="1" EventTime="2007-05-01T09:59:49.000">
```

```
<ns:SectionAddedNotice sectionUuid="5b035ebf64f0417fadbad47c114299ce" beforeUuid=""/>
</ns:Notice>

<!-- when a section is deleted -->
<ns:Notice EventType="SectionDeleted" MachineName="K2-1" DatasetName="V:" GroupName="default"
MovieName="~scratchclip" ModifyMask="2" EventTime="2007-05-01T10:24:33.000">
<ns:SectionDeletedNotice sectionUuid="c428b9d036fb43dba5ca5658c7c518d7" duration="0"/>
</ns:Notice>

<!-- when a section is moved -->
<!-- TODO: EventType="SectionMoved" -->

<!-- when a section is modified -->
<!-- TODO: EventType="SectionModified" -->

<!-- EDIT EVENTS -->
<!-- when an edit is added (e.g. insert play event) -->
<ns:Notice EventType="EditAdded" MachineName="K2-1" DatasetName="V:" GroupName="default"
MovieName="~scratchclip" ModifyMask="4" EventTime="2007-05-01T10:31:42.000">
<ns:EditAddedNotice editType="PlayEvent" editLayer="0" sectionUuid="3ceee71f4e9c4c1284d958afb176f102"
editUuid="3585778b3f684c8b951d52fefeed0f69" beforeUuid="" duration="1032"/>
</ns:Notice>


<!-- when an edit is deleted (e.g. delete a play event) -->
<ns:Notice EventType="EditDeleted" MachineName="K2-1" DatasetName="V:" GroupName="default"
MovieName="~scratchclip" ModifyMask="8" EventTime="2007-05-01T10:33:53.000">
<ns:EditDeletedNotice sectionUuid="3ceee71f4e9c4c1284d958afb176f102" editLayer="0"
editUuid="3585778b3f684c8b951d52fefeed0f69" duration="1032"/>
</ns:Notice>

<!-- when an edit is moved -->
<!-- TODO: EventType="EditMoved" -->

<!-- when an edit is modified -->
<!-- TODO: EventType="EditModified" -->

<!-- when an edit marks change -->
<!-- TODO: EventType="EditMarks" -->

<!-- when the current edit changes -->
<!-- TODO: EventType="CurrentEditChanged" -->

</ns:Data>
</XmlChanges>
```

Appendix E: Error Codes

Error codes in the logs are usually displayed in a hexadecimal format like the error codes below. Any error code that starts with 0x8004 prefix is an error code defined by Grass Valley.

Other error codes are most often defined by Microsoft. To get more information about Microsoft error codes visit this link:
http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BMicrosoft%20error%20codes%5D%5D

A header file containing these error codes can be found here:
Grass Valley K2 error codes
http://www.gvgdevelopers.com/K2DevGuide/K2DevGuide.html#%5B%5BGrass%20Valley%20K2%20error%20codes%5D%5D

System errors:

Code	**Error**
0x80041000	SYS_E_GENERAL_ERROR
0x80041001	SYS_E_FILENOTOPEN
0x80041002	SYS_E_FILEALREADYOPEN
0x80041003	SYS_E_FILE
0x80041004	SYS_E_UNKNOWN_FORMAT

Timeline Server errors:

Code	**Error**
0x80043001	TMSSRV_E_CHANNEL_NOT_ALLOCATED
0x80043002	TMSSRV_E_INVALID_RESOURCE_TYPE
0x80043003	TMSSRV_E_INVALID_CONTEXT_CHANGE
0x80043004	TMSSRV_E_THREAD_STARTUP_FAILED
0x80043005	TMSSRV_E_RIFF_CREATE
0x80043006	TMSSRV_E_RIFF_READ
0x80043007	TMSSRV_E_RIFF_WRITE
0x80043008	TMSSRV_E_RIFF_ASCEND
0x80043009	TMSSRV_E_RIFF_DESCEND
0x8004300A	TMSSRV_E_RIFF_OPEN
0x8004300B	TMSSRV_E_RIFF_MEM_ACCESS
0x8004300C	TMSSRV_E_RIFF_BAD_SIZE
0x8004300D	TMSSRV_E_RIFF_SEEK
0x8004300E	TMSSRV_E_INVALID_RESOURCE_PROPERTY
0x8004300F	TMSSRV_E_INVALID_RESOURCE_OP
0x80043010	TMSSRV_E_UNKNOWN_VOLUME_NAME
0x80043011	TMSSRV_E_IPM_TIMEOUT
0x80043012	TMSSRV_E_RTS_FAILURE
0x80043013	TMSSRV_E_VIDEO_FORMAT
0x80043014	TMSSRV_E_VIDEO_COMPRESSION
0x80043015	TMSSRV_E_RTS_SHMEM
0x80043016	TMSSRV_E_BAD_FILE_RANGE
0x80043017	TMSSRV_E_INVALID_CHAN_PROPERTY
0x80043018	TMSSRV_E_INVALID_CHAN_OP
0x80043019	TMSSRV_E_TOO_CLOSE_TO_RECORD
0x8004301A	TMSSRV_E_CHAN_OFFLINE
0x8004301B	TMSSRV_E_FILE_OPEN_FAILED
0x8004301C	TMSSRV_E_FILE_CREATE_FAILED

Transfer Queue & Transfer Session errors:

Code	**Error**
0x80044002	XFRSRV_E_REMOVE_TRANSFERRING_ITEM
0x80044003	XFRSRV_E_LOAD_XML
0x80044005	XFRSRV_E_CREATE_MMF
0x80044006	XFRSRV_E_MAP_VIEW
0x80044007	XFRSRV_E_QUEUE_THREAD_START
0x80044008	XFRSRV_E_ABORT_ITEM
0x80044009	XFRSRV_E_ITEM_NOT_FOUND
0x8004400A	XFRSRV_E_UPDATE_ABORTED_ITEM
0x8004400B	XFRSRV_E_INVALID_NAME

Transfer session & server errors:

Code	**Error**
0x80044c01	E_XFER_NOT_SUPPORTED
0x80044c02	E_XFER_ABORTED_NO_PROGRESS
0x80044c03	E_NO_SYS_RESOURCES
0x80044c04	E_XFER_SESSION_LOAD_ERROR
0x80044c05	E_XFER_BAD_UML_FORM
0x80044c06	E_XFER_CTL_ABORTED
0x80044f01	E_XFER_REACH_QUEUE_CAPACITY

Transfer movie data provider errors:

Code	**Error**
0x80044e01	E_MDP_GENERAL_FAILURE
0x80044e02	E_MDP_BAD_UML_FORM
0x80044e03	E_MDP_MOVIE_EXISTS
0x80044e04	E_MDP_BAD_MOVIE
0x80044e05	E_MDP_XFER_NOT_DEF
0x80044e06	E_MDP_MDS_ERROR
0x80044e07	E_MDP_MOVIE_NOT_EXIST
0x80044e08	E_MDP_NO_MOVIE_PERMISSION
0x80044e09	E_MDP_NO_BIN_PERMISSION
0x80044e0A	E_MDP_DIRTY_CLIP

Transfer filter subsystem errors:

Code	**Error**
0x80044D01	E_FLT_GENERAL_FAILURE
0x80044D02	E_FLT_UMF_FORM
0x80044D03	E_FLT_BAD_STREAM
0x80044D04	E_FLT_BAD_MEDIATYPE
0x80044D05	E_FLT_BAD_UML_FORM
0x80044D06	E_FLT_BAD_MOVIE
0x80044D07	E_FLT_STRM_TOO_SHORT
0x80044D08	E_FLT_NET_ERROR
0x80044D09	E_FLT_UNSUPPORTED_VIDEO_RESOLUTION
0x80044D0A	E_FLT_UNSUPPORTED_AUDIO_SAMPLING
0x80044D0B	E_FLT_UNSUPPORTED_AUDIO_FORMAT
0x80044D0C	E_FLT_UNSUPPORTED_AUDIO_SAMPLESIZE
0x80044D0D	E_INPROG_MOVIE_TIMEOUT
0x80044D0E	E_FLT_STRM_ABORT
0x80044D0F	E_FLT_SRL_NOT_ALLOWED
0x80044D10	E_FLT_OPEN_GOP
0x80044D11	E_FLT_BOTTOM_FIELD_FIRST
0x80044D12	E_FLT_ILLEGAL_MEDIA_STATE

Transfer MXF filter errors:

Code	**Error**
0x80044b01	E_MXF_GENERAL_FAILURE
0x80044b02	E_MXF_CONFORMANCE_FAILURE
0x80044b03	E_MXF_KLVPARSE_FAILURE
0x80044b04	E_MXF_UNSUPPORTED_VID_FORMAT
0x80044b05	E_MXF_HDR_CONSISTENCY_FAILURE
0x80044b06	E_MXF_RUNINS_NOT_SUPPORTED
0x80044b07	E_MXF_UNSUPPORTED_OP_FORMAT
0x80044b08	E_MXF_DICTIONARY_LOOKUP_ERROR
0x80044b09	E_MXF_MISSING_ESSENCE_DATA
0x80044b0A	E_MXF_OPEN_HEADER_FAILURE

AppServer errors:

Code	**Error**
0x80045001	APPSRVDOTNET_E_STOLEN_CHANNEL
0x80045002	APPSRVDOTNET_E_CHANNEL_IN_USE
0x80045003	APPSRVDOTNET_E_SUITE_IN_USE
0x80045004	APPSRVDOTNET_E_APPSERVICE_OFFLINE

Movie Server errors:

Code	**Error**
0x80045201	MOVSRV_E_FIND_END
0x80045202	MOVSRV_E_COPY_MOVIE_COMPLETED
0x80045203	MOVSRV_E_DUPLICATE_NAME
0x80045204	MOVSRV_E_ALREADY_OPEN
0x80045211	MOVSRV_E_NO_TOKENS_AVAILABLE
0x80045212	MOVSRV_E_OBJECT_READONLY
0x80045213	MOVSRV_E_MOVIE_LOCKED
0x80045214	MOVSRV_E_NOT_CONTROLLER
0x80045215	MOVSRV_E_COPY_MOVIE_DENIED
0x80045216	MOVSRV_E_SHARED_MEDIA
0x80045217	MOVSRV_E_MEDIA_FILESYSTEM_FULL
0x80045218	MOVSRV_E_BAD_CMF_FILE_LEN
0x80045221	MOVSRV_E_INVALID_INPUT
0x80045222	MOVSRV_E_BAD_NAME_STRING
0x80045223	MOVSRV_E_STRING_TOO_LONG
0x80045224	MOVSRV_E_BAD_TOKEN
0x80045225	MOVSRV_E_NOT_CLIP
0x80045226	MOVSRV_E_OPERATION_NOT_ALLOWED
0x80045231	MOVSRV_E_SOME_NOTIFICATION_DATA_DISCARDED
0x80045232	MOVSRV_E_INVALID_NOTIFICATION
0x80045241	MOVSRV_E_MUTEX_TIMEOUT
0x80045242	MOVSRV_E_SLOT_TIMEOUT
0x80045243	MOVSRV_E_DB_SEARCH_FAILED
0x80045244	MOVSRV_E_INVALID_DATA
0x80045245	MOVSRV_E_SYSTEM_ERROR
0x80045246	MOVSRV_E_BAD_SLOT
0x80045247	MOVSRV_E_SLOT_LOCK_ERROR
0x80045250	MOVSRV_E_BAD_POINTER_ARGUMENT
0x80045251	MOVSRV_E_MUST_BE_LOCAL_CONNECTION
0x80045252	MOVSRV_E_DB_SERVER_CONNECTION_FAILED
0x80045260	MOVSRV_E_INVALID_SECTION
0x80045261	MOVSRV_E_INVALID_EDIT
0x80045262	MOVSRV_E_INVALID_SECTION_PROPERTY
0x80045263	MOVSRV_E_INVALID_EDIT_PROPERTY
0x80045264	MOVSRV_E_INVALID_SECTION_OP
0x80045265	MOVSRV_E_BAD_URI_STRING

0x80045266	MOVSRV_E_BAD_URI_TYPE
0x80045267	MOVSRV_E_INVALID_ASSET_OP
0x80045268	MOVSRV_E_INVALID_CTX_CHANGE
0x80045269	MOVSRV_E_BAD_SECTION_TIMECODE
0x8004526A	MOVSRV_E_NO_TRACKS
0x8004526B	MOVSRV_E_MIN_SECTION_LIMIT
0x8004526C	MOVSRV_E_MAX_SECTION_LIMIT
0x8004526D	MOVSRV_E_MAX_EDIT_LIMIT
0x8004526E	MOVSRV_E_MIN_FIELDS_LIMIT
0x8004526F	MOVSRV_E_PERMISSION_DENIED
0x80045270	MOVSRV_E_TIMECODE_NOT_FOUND

Configuration Server errors:

Code	**Error**
0x80046201	CFGSRV_E_DATABASE_CONNECTION
0x80046202	CFGSRV_E_RTS_CONNECTION
0x80046203	CFGSRV_E_REMOTEHOST_NOT_EXIST
0x80046204	CFGSRV_E_RTS_IPM_CONNECT_ERROR
0x80046205	CFGSRV_E_RTS_IPM_THREAD_STARTUP
0x80046206	CFGSRV_E_RTS_IPM_THREAD_SHUTDOWN
0x80046207	CFGSRV_E_RTS_IPM_THREAD_TIMEOUT
0x80046208	CFGSRV_E_RTS_INVALID_DATA
0x80046209	CFGSRV_E_UNKNOWN_BOARD_TYPE
0x8004620A	CFGSRV_E_COMMIT_IN_PROGRESS
0x8004620B	CFGSRV_E_FILE_WRITE_FAILURE

Channel Status errors:

Code	**Error**
0x80048001	CHSRV_E_STRING_TOO_LONG
0x80048002	CHSRV_E_BAD_TEXT_INDEX
0x80048003	CHSRV_E_THUMBNAIL_TOO_BIG
0x80048004	CHSRV_E_EMPTY_STATUS_QUERY
0x80048005	CHSRV_E_UNKNOWN_STATUS_QUERY
0x80048006	CHSRV_E_UNKNOWN_CHAN_NAME
0x80048007	CHSRV_E_MMF_MUTEX_TIMEOUT

MediaMgr Server errors:

Code	**Error**
0x80049001	MMSRV_E_INVALID_INPUT
0x80049002	MMSRV_E_BIN_EXIST
0x80049003	MMSRV_E_BIN_NOT_EXIST
0x80049004	MMSRV_E_REMOTEHOST_NOT_EXIST
0x80049005	MMSRV_E_UNKNOWN_URI_FORMAT

Control Server errors:

Code	**Error**
0x8004a001	CTLSRV_E_NOT_CONNECTED
0x8004a002	CTLSRV_E_NOT_OPEN
0x8004a003	CTLSRV_E_NO_CLIP_LOADED
0x8004a004	CTLSRV_E_NO_UNIQUE_NAME
0x8004a005	CTLSRV_E_UNKNOWN_URI_FORMAT
0x8004a006	CTLSRV_E_NEGATIVE_OFFSET
0x8004a007	CTLSRV_E_UNKNOWN_CTX_TRIGGER
0x8004a008	CTLSRV_E_CHAN_TYPE
0x8004a009	CTLSRV_E_NO_PORT_CONTEXT
0x8004a00A	CTLSRV_E_RTS_SHMEM
0x8004a00B	CTLSRV_E_UNKNOWN_CTX_CONDITION
0x8004a00C	CTLSRV_E_TIMECODE_NOT_FOUND

0x8004a00D	CTLSRV_E_BAD_CTX_CHANGE

AppCenter errors:

Code	**Error**
0x8004b001	FRMWK_E_NOT_OPEN
0x8004b002	FRMWK_E_STARTUP
0x8004b003	FRMWK_E_ALREADY_OPEN

Quality of Service Manager errors:

Code	**Error**
0x8004d201	QOSSRV_E_STRING_TOO_LONG
0x8004d202	QOSSRV_E_MMF_MUTEX_TIMEOUT
0x8004d203	QOSSRV_E_CLIENT_EXISTS
0x8004d204	QOSSRV_E_CLIENT_NOT_FOUND
0x8004d205	QOSSRV_E_CLIENTS_FULL
0x8004d206	QOSSRV_E_INVALID_CLIENT_INDEX
0x8004d207	QOSSRV_E_VOLQOSACCESS_NOT_FOUND
0x8004d208	QOSSRV_E_QOS_NOT_SUPPORTED

Error chain context errors

Code	**Error**
0x8004ffxx	These are error messages that are used to chain other error messages together. They are used internally for creating a stack trace of all calls in an error.