Matchmaker SDK

Prepared by:
Alexander Scholz

Version:
1.0

Component History

Version	Component

Document History

Document name and Version	Author	Reviewed By	Release Date
MatchmakerSDK V 1.0	J. Price, A. Scholz		10/00

Introduction to Matchmaker 1.0

The Matchmaker is an application that matches two or more players to play a multiplayer game on EA.com.

The Matchmaker was developed using the Kesmai Matchmaking System (KMS) engine. KMS provides basic player chat with a robust set of matchmaking features integrated into the EA.com Enterprise Infrastructure. The system features a standard matchmaking Graphical User Interface, complete with game lobbies, game rooms, game settings, and chat.

Features

- Launch mechanics for Web-based games
- Peer-to-peer and host-client support (2-n players).
- Scalable lobby support
- Separate lobby and room chat
- Configurable game and player settings (passable into a game)
- Administrative features such as “Boot and Ban” from game sessions
- Room “Creator” features (moderation)

Developer Needs

EA.com provides game developers the Game Control interface to communicate from a game control host to a game control. The Game Control can be written in Visual Basic, Visual C++ or any language that supports creating ActiveX interfaces. Game developers create a game control to customize the Matchmaker for their game. The game control uses the Active X interface to collect game and player specific settings from the user and then passes those settings to the game so it will launch with the proper multiplayer configuration.

Frequently Asked Questions

Can I download a developer's kit and what does it consist of?

The Matchmaker SDK is available at
http://www.bos.ea.com/gametech/capabilities/matchmaker.

At present the developer's kit consists of a sample game control and related libraries, a test container and documentation. The test container is much like the core Matchmaker client, except that it does not require any type of server connection. It initiates a game control, allows you to display its parts, and provides first-level testing of the control–hopefully allowing quicker development by avoiding firewall and connectivity issues.

Do you have a sample game control I can start with and extend?

Yes, the developer's kit contains a sample game control. It is an ATL (Active Template Library) ActiveX control. We chose that development route in an attempt to make the control smaller with fewer dependencies.

What are the rules for what is and what is not allowed in a game control?

The EA game control interface (IEAGameControl) determines most of what you can and can not do with the Matchmaker. This interface is the means of communication between a game control and the Matchmaker core client.

The game control serves the purpose of customizing the look of the Matchmaker, determining game settings, determining player settings, and launching the game. The game controls can pretty much do anything that makes sense for the game. Since they are a very game-specific application, they have a lot of capabilities and can be designed to do most everything that a Windows application can do. With that in mind, we have to be careful not to let them get too big, as many of our customers will be downloading them.

How do I test the game when I do not have a game control written yet?

By using the XML game launch file or Web page parameters. It is not necessarily obvious that you do not need the Matchmaker and game control to start testing the game brick. Many of the EA.com games are going to be browser games in which the game brick will be embedded in a Web page. In most cases you can set up hard-coded parameters in your HTML <object> or <embed> tags. You can launch your own Web page and test that way. A Web based game can also read from the XML game launch file, though that has not been a standard for games set in a browser.

CD-ROM games can be launched from the CD, using parameters read from the XML game launch file. With a game like NHL 2001, which is a CD-ROM game, game settings are passed via an XML file which is written to the game's main directory. We can create

test XML files with settings that the games can read, so that you do not need the rest of the Matchmaker to launch the game. From the XML file you get all the relevant information you need to play the game: IP addresses for all of the other players, as well as game settings and player settings.

You can create an XML file, basically a text file, in Microsoft Notepad or another editor. You do not even need a browser to launch a CD-ROM game. If the file is in the right place, your game should find the file and parse it.

We have a lot of flexibility in how we design game controls, so we're not limited to only those two methods, but they are the current standards for launching games.

How do I install and run the debug game control so that I can verify that it works?

The game control is a .dll. The browser instantiates or creates an instance of it, brings the library into memory, and then starts calling the code. It is an object-oriented system, so a Matchmaker host is created first. Then the Matchmaker creates a game control object. Both the Matchmaker and the game control are ActiveX controls. The way Windows finds the .dll required to instantiate the objects is via a GUID (Globally Unique Identifier). Both the Matchmaker's and the game control's GUIDs are in the Web page in which the Matchmaker is embedded.

The HTML <object> and <embed> tags identify the Matchmaker ActiveX control GUID, and then the game control GUID is passed in to the Matchmaker as a parameter. When the browser goes to substantiate the Matchmaker, Windows uses the fact that these GUID's are registered. You can register your game control by using REGSVR32.exe.

REGSVR32.exe resides in the system directory on all Windows computers. To register your .dll on a Windows system, select "Run" from the Start menu and then type "regsvr32" followed by the location and name of your .dll. Once that is done, anything using that GUID will be able to find that .dll, as long as it has not been moved. Once the game control .dll is registered, use the test container or the Matchmaker to test the game control. If you are using a test container you do not have to use a server. If you are using the Matchmaker game host you have to be connected to a server.

The Microsoft Visual C++ development environmentl registers the game control .dll after every compile for developers. When EA.com users go to play your game, the game control is downloaded and registered by ESD (Electronic Software Distribution).

Once I have a game control, how do I add it to Matchmaker for testing?

There is a GUID that is passed in as a parameter to the Matchmaker in the Web page in which Matchmaker is embedded. Assuming the .dll is registered properly, when you open

the page, the GUID parameter gets passed in to the Matchmaker, and the Matchmaker attempts to instantiate the control.

How do I get Matchmaker up and running so I can see the whole system work?

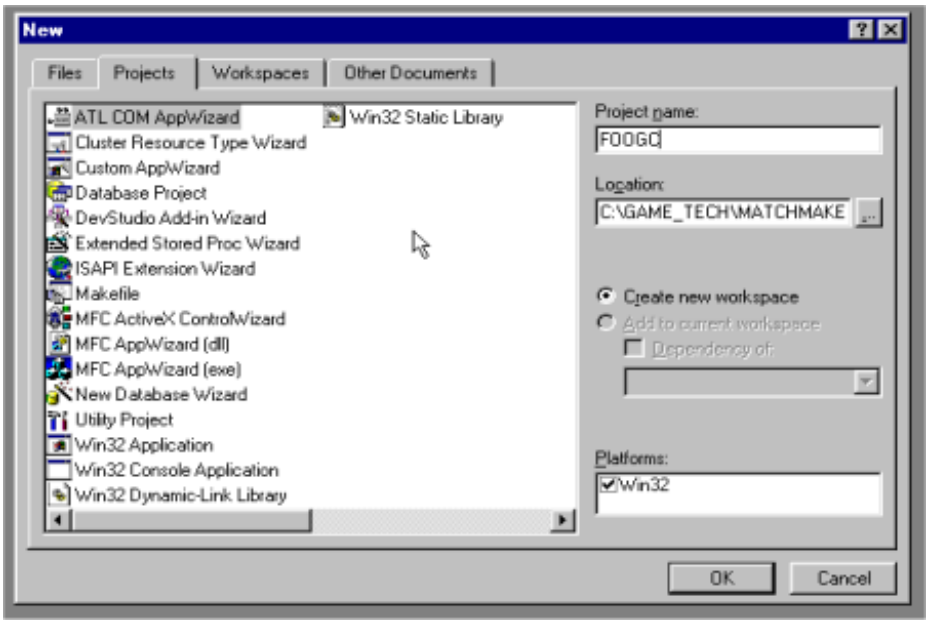
Developers should be working with their GIA (Game Integration Architect). If they do not have a GIA, they can work with the Matchmaker client team to figure out what needs to be done. There are may details that must be taken into consideration, including configuration file and database changes, as well as web page creation.

Matchmaker Game Control – Getting Started

This section provides information on creating a basic game control. This tutorial will walk you through the creation of a Matchmaker game control for a hypothetical game, Foo. The creation of a new game control will parallel this entirely.

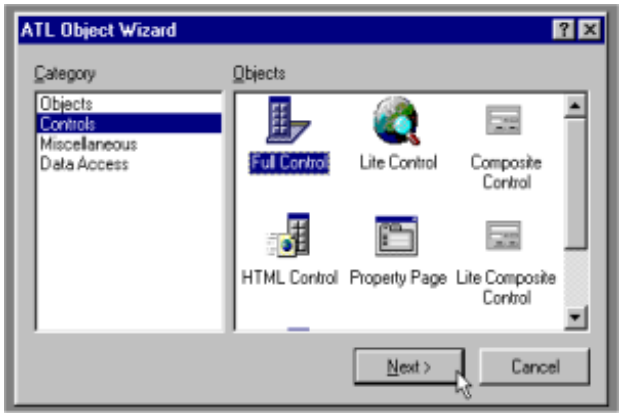
Steps to Create a Game Control

1) Open MSVC (Microsoft Visual C++ development environment).
2) Create a new **ATL COM AppWizard** Project.
    a. Choose **New** from the **File** menu.
    b. Make sure you are looking at the **Projects** tab.
    c. Select **ATL COM AppWizard**.



    d. Set the location you would like the new project created in the **Location** edit box on the right hand side.
    e. Name your project in the **Project Name** box. In this example, because the game is named “Foo”, we are naming the project FOOGC (for Foo Game Control).
    f. Click the **OK** button.
    g. You should now be in the ATL AppWizard. The defaults (Server type DLL, nothing else checked) are correct. Click the **Finish** button.
    h. A dialog will pop up showing you the project settings. Nod knowingly and click the **OK** button.
3) You should now be in class view. If you are not, click the **ClassView** tab in the Workspace.
4) We now need to create the base COM object.
    a. Right click on the class name in class view (In this example: “Foo classes”).
    b. Select the **New ATL Object** menu item.
    c. In **Category**, select **Controls.**

d. In **Objects,** select **Full Control**.



e. Click the **Next** button.

f. Enter the “Short Name” for the new object. This name will define an interface, which we will promptly ignore. It will also be the string class name that may be used to instantiate the control; in the Matchmaker this is also ignored. Finally this will name the class in which we will implement the Game Control interface. In our case we will use the short name “FooGCImpl” for Foo Game Control Implementation.

g. Select the **Miscellaneous** tab.

h. In the bottom right hand corner, make sure the **Windowed Only** checkbox is checked. It is not checked by default.

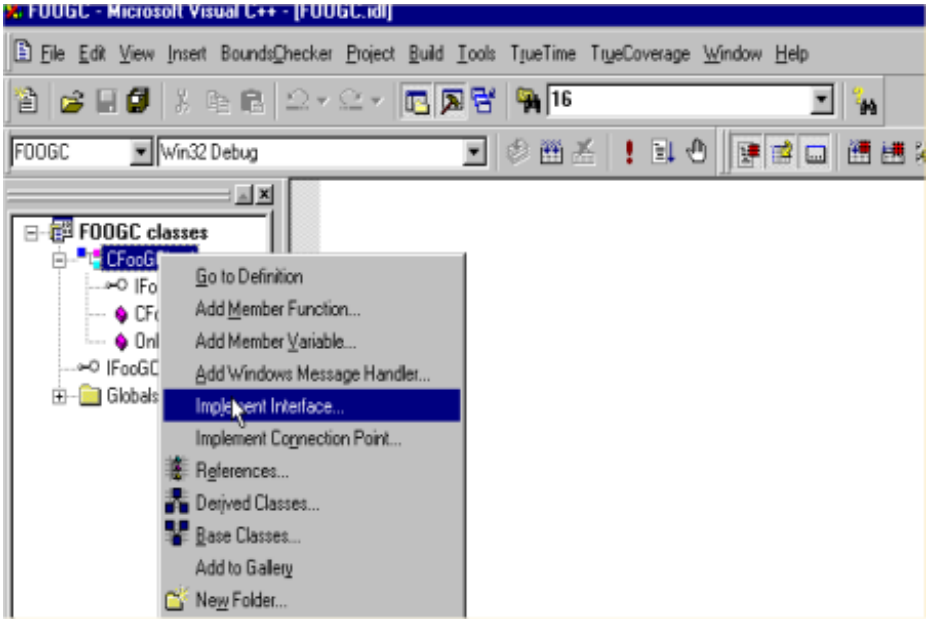
i. Click the **OK** button.

5) You should now be once again back in class view. If you are not, click the **ClassView** tab in the Workspace.

6) Now we are going to let the wizards implement the game control interface for us.

a. Expand the class list. (Click the **Plus (+)** sign next to “<ProjectName> classes.)

b. You should now see a class with the short name you selected in step 4, preceded by the letter “C.” You will also see an interface of the same name next to a spoon icon. This is the interface we are going to ignore.



c. Right click on the class name (In this example: “CFooGCImpl”)

d. Select the **Implement Interface…** menu item.

e. A warning should pop up telling you it could not find the type library for this project… This is fine. Click the **OK** button.

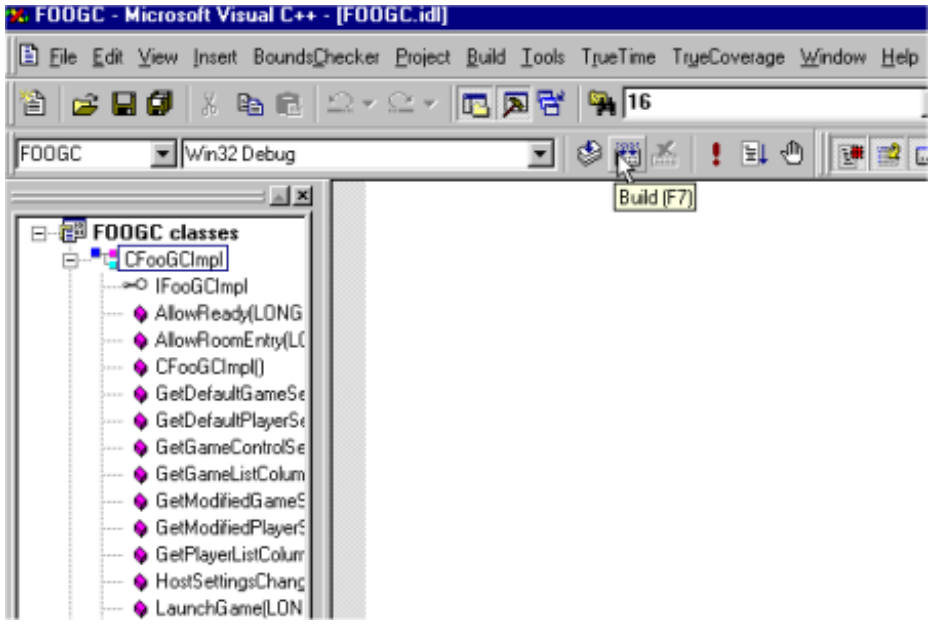
f. A new dialog box should appear allowing you to browse type libraries. On the right hand side you should click the **Browse** button.

g. A **File Open** dialog box will appear. Navigate to the “common” directory in the SDK directory. Select the file “EAGC” and click the **Open** button.

h. You should now be in the **Implement Interface** dialog box with two interfaces to choose from (IEAGameControl and IEAGameControlHost). Check the box next to IEAGameControl and click the **OK** button.

i. Your implementation class will now have the methods of the IEAGameControl interface.

7) Try compiling to verify everything has worked up until now. Assuming it compiled, you should now have a game control that does absolutely nothing useful☺. How you implement the rest is very flexible and can be done in many ways.



8) For the other methods look in the Foo example.
9) For each **Settings View** (tab) you want to display you will probably want a dialog to represent it. These steps show you one way to create these dialogs:
    a. Select the **Insert** menu, then click the **New ATL Object…** menu item.
    b. Select Category: **Miscellaneous**, Objects: **Dialog**.
    c. Click the **Next** button.
    d. Type in a short name for the new dialog (In this example: FooPlayerSettingsDlg).
    e. Click the **OK** button.
    f. In the .dlg editor set the dialog properties to be of Style: **child**, with no border. You will probably want to delete the default buttons as well. Remember to disable or delete their handlers also.
    g. Setup the dialog how you would like it to be. Look at the DlgDrawer lib documentation and the Foo example to see how to get skinning somewhat automatically (NOTE: to use the dlg drawer you will need to link to dlgdrawer.lib, zlib.lib and libpng.lib).
10) Finally, the game control is responsible for packing and unpacking data into a BLOB (Binary Larger Object) for transporting across the network. The game control host expects data to be packed into a SafeArray in a variant of type VT_ARRAY|VT_UI1. (Refer to the Foo game control and DEADLib (Data Encoding And Decoding) documentation for more details).

DlgDrawer API

Files Included in the DlgDrawerAPI ZIPfile

- Collections.h
- EADataObject.h
- eammgraphics.h
- eammutility.h
- EAWindowMessages.h
- ImageManager.h
- ReleaseUMinDependency
- TmpltDlgDrawer.h
- png.h
- pngsupport.h
- libpng.lib
- DlgDrawer.h
- DlgDrawer.lib
- zlib.h
- zlib.lib

Note: This is a short-lived library. Shortly both the Matchmaker and the game control will replace this with a common library that will be shared. Usage will probably change, but only slightly.

DlgDrawer Overview

DlgDrawer is designed to ease the skinning of game controls. It is a base class that you derive your CWindow-based dialogs from. By ‘chaining’ the Windows messages up to DlgDrawer the painting and image placement will be largely automatic. DlgDrawer can also be set to automatically size buttons to fit the images (if available) and handle scrolling.

DlgDrawer is largely (99%) derived from the code used in the Matchmaker, so the look should be almost identical.

DlgDrawer Programming Guide

This is MSVC/WIN32 only.
The primary class is DlgDrawer.
To use DlgDrawer In your CWindow derived class, you should derive from DlgDrawer as such:

First derive from it:

```
class CFooGameSettingsDlg :
public CAxDialogImpl<CFooGameSettingsDlg>,
public DlgDrawer
```

Pass a handle to the CWindow(this), initialized structures (FontSet, Color Set and Image-Manager [defined in collections.h and ImageManager.h]) to the DlgDrawer constructor (see the foo example for creating a game control). The final two parameters are both optional and default to false. They are bAutoSizeButtons (which, if set to true, will search for buttons owned by the Dialog and size them for skinning) and bAutoScroll (which, if set true, will add scroll bars and scroll the children windows for you).

```
CFooGameSettingsDlg(ISettingsChanged *ptrISC, FontSet *pFontSet, ColorSet *pColorSet, cEAIm-
ageManager *pImageManager) : DlgDrawer(this, pFontSet, pColorSet, pImageManager, true, true),…
```

Chain messages to DlgDrawer in the message map.

NOTE: position the CHAIN_MSG_MAP at the end of the message map for scrolling to work properly. Be careful to reposition it at the end if you use wizards to add handlers as they tend to add at the end.

```
BEGIN_MSG_MAP(CFooGameSettingsDlg)
MESSAGE_HANDLER(WM_INITDIALOG, OnInitDialog)
COMMAND_ID_HANDLER(IDOK, OnOK)
COMMAND_ID_HANDLER(IDCANCEL, OnCancel)
MESSAGE_HANDLER(WM_SIZE, OnSize)
CHAIN_MSG_MAP(DlgDrawer)
END_MSG_MAP()
```

Finally, in all the message handlers you use, set the bHandled reference parameter to FALSE.

```
LRESULT OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled)
{
        bHandled = false;
        return 1;  // Let the system set the focus
}
```

KickApp API

Files Included in the Kick Application API ZIPfile

kickapp.lib
kickapp.h

KickApp API Overview

KickApp is designed as a mechanism to allow for game pre-launch and communication between the game and the Matchmaker.

The game may be running in the background at the same time the user is browsing through the Matchmaker. When it is time to play, the game control will call a kickapp command to wake the game up or launch the game if it is not already running. When the game is complete, kickapp will signal an event letting the Matchmaker know that the game has ended.

KickApp Client Interfaces

HANDLE Play()			
Description	Tells kicking to play the game. If the game is already running, it will wake it up. If the game is not yet running, it will execute the game as well.		
Prerequisites			
Syntax	HANDLE Play(const TCHAR *serviceName, const TCHAR *gameName, const TCHAR *parameters, const TCHAR *workingDirectory = NULL);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
const TCHAR *serviceName	In	TCHAR	Required name of the game host service.
const TCHAR *gameName	In	TCHAR	Required path and name of the game executable file, including any parameters required by the game.
const TCHAR *parameters	In	TCHAR	Required name of the XML file.
const TCHAR *workingDirectory	In	TCHAR	Required location for game installation.
Error Messages			
Return Values	Handle	Not referenced 0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

KickApp code sample

```
KickApp     m_KickApp;

m_KickApp.Play(_T("FIFA2001"), cmdLine.c_str(),
strXMLJustFileName.c_str(), m_strGameDir.c_str());
```

XML Write API

Files Included in the XML WriteAPI ZIPfile

xmlwrite.lib
xmlwrite.h
xmllaunch.h

XML Write API Overview

The xmlwrite library is used to write the game settings as specified in the Matchmaker to an XML file. These settings are read by the game at launch.

XML Client Interfaces

XMLCreate()			
Description	Creates the XML file.		
Prerequisites			
Syntax	bool XMLCreate(TCHAR *fileName);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
fileName	In	TCHAR	Creates the xml file.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

XMLWriteComplete()				
Description	Closes the XML file and writes the ending tag. For the purposes of writing an XML launch file, the end tag is “TAG_FILEID” which is “MMLAUNCH” .			
Prerequisites				
Syntax	bool XMLWriteComplete(TCHAR *tag = NULL);			
Arguments		**In, Out, In/Out**	**Type**	**Description**
tag		In	TCHAR	Required XML launch file end tag.
Error Messages				
Return Values			0 = True on success. 1 = HRESULT error code on failure.	
Exceptions				
Notes				

MMLaunchCreate()			
Description	Writes the XML file header description tags and information.		
Prerequisites			
Syntax	bool MMLaunchCreate(TCHAR *xmlName, TCHAR *docType, TCHAR *mmVersion, TCHAR *gameVersion);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
xmlName	In	TCHAR	“TAG_FILEID” which is “MMLAUNCH”.
docType	In	TCHAR	"MMLAUNCH SYSTEM” \"MMLaunch.dtd\".
mmVersion	In	TCHAR	Matchmaker version (The string value obtained from the Matchmaker API call GetGameControlHostVersion).
gameVersion	In	TCHAR	Game version (hard coded to 1.2).
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

HostInfo ()

Description	Writes out the game host tag information.
Prerequisites	
Syntax	bool HostInfo(TCHAR *attr1, TCHAR *attr2, TCHAR *name, TCHAR *address, int indent = 1);

Arguments	In, Out, In/Out	Type	Description
attr1	In	TCHAR	“ATTR_CLIENTTYPE”.
attr2	In	TCHAR	“ATTR_CLIENTTYPE_LOCAL” or “ATTR_CLIENTTYPE_REMOTE”, depending on who is the game creator, “ATTR_CLIENTTYPE_LOCAL” if the PC is the game creator, “ATTR_CLIENTTYPE_REMOTE’ of the PC is a game client.
name	In	TCHAR	Name of the game creator (EA login name).
address	In	TCHAR	IP address of the game creator.
indent	In	Int	Number of tabs (4 spaces) to indent the host tag data.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

ClientInfo()

Description	Writes out the game client tag inforomation.
Prerequisites	
Syntax	bool ClientInfo(TCHAR *attr1, TCHAR *attr2, TCHAR *name, TCHAR *address, int indent = 1);

Arguments	In, Out, In/Out	Type	Description
attr1	In	TCHAR	“ATTR_CLIENTTYPE”.
attr2	In	TCHAR	“ATTR_CLIENTTYPE_LOCAL” or “ATTR_CLIENTTYPE_REMOTE”, depending on who is game creator, “ATTR_CLIENTTYPE_LOCAL” if the PC is game creator, “ATTR_CLIENTTYPE_REMOTE’ if the PC is a game client.
name	In	TCHAR	Name of the game creator (EA login name).
address	In	TCHAR	IP address of the game creator.
indent	In	Int	Number of tabs (4 spaces) to indent the host tag data.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

GameSettings()

Description	Writes the game setting tag information.
Prerequisites	

Syntax	bool GameSettings(int id, TCHAR *value, int indent = 1);
	bool GameSettings(TCHAR* id, TCHAR *value, int indent = 1);
	bool GameSettings(int id, int value, int indent = 1);
	bool GameSettings(TCHAR* id, int value, int indent = 1);

Arguments	In, Out, In/Out	Type	Description
id	In	Int or TCHAR	Game setting ID.
value	In	Int or TCHAR	Game setting value.
indent	In	Int	Number of tabs (4 spaces) to indent the host tag data.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

// write player settings

PlayerSettings()			
Description	Writes the player setting tag info.		
Prerequisites			
Syntax	bool PlayerSettings(TCHAR *name, TCHAR *id, TCHAR *value, int indent = 1);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
name	In	TCHAR	Name of the player (EA login name).
id	In	TCHAR	Player setting id.
value	In	TCHAR	Player setting value.
indent	In	Int	Number of tabs (tab = 4 spaces) to indent the host tag data.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

Helper functions

```
// write tagged data (output = indent spaces <tag-
name>data</tagname>linefeed)
```

WriteTaggedData()				
Description	Writes tagged TCHAR data.			
Prerequisites				
Syntax	bool WriteTaggedData(TCHAR *tag, TCHAR *data, int indent = 0);			
Arguments		**In, Out, In/Out**	**Type**	**Description**
tag		In	TCHAR	Tag name.
data		In	TCHAR	Tag data.
indent		In	Int	Number of tabs (tab = 4 spaces) to indent the host tag data.
Error Messages				
Return Values			0 = True on success. 1 = HRESULT error code on failure.	
Exceptions				
Notes				

WriteTaggedData()			
Description	Writes tagged integer data.		
Prerequisites			
Syntax	bool WriteTaggedData(TCHAR *tag, TCHAR *data, int indent = 0);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
tag	In	TCHAR	Tag name.
data	In	TCHAR	Tag data.
indent	In	Int	Number of tabs (tab = 4 spaces) to indent the host tag data.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

WriteAttribute()			
Description	Writes the XML file header description tags and information.		
Prerequisites			
Syntax	bool WriteAttribute(TCHAR *tag, TCHAR *attr1, TCHAR *attr2, int indent = 0);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
tag	In	TCHAR	Tag name.
attr1	In	TCHAR	Attribute name.
attr2	In	TCHAR	Attribute value.
indent	In	Int	Number of tabs (tab = 4 spaces) to indent the host tag data.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

BeginTag()			
Description	Writes a begin tag, “<tagname>”.		
Prerequisites			
Syntax	bool BeginTag(TCHAR *tag, bool newline, int indent = 0);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
tag	In	TCHAR	Tag name.
newline	In	BOOL	True of you want a line feed.
indent	In	Int	Number of tabs (tab = 4 spaces) to indent the host tag data.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

EndTag()			
Description	Writes an end tag, “</tagname>”.		
Prerequisites			
Syntax	bool EndTag(TCHAR *tag, bool newline, int indent = 0);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
tag	In	TCHAR	Tag name.
newline	In	BOOL	True of you want a line feed.
indent	In	Int	Number of tabs (tab = 4 spaces) to indent the host tag data.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

WriteData()			
Description	Writes TCHAR ‘data’ to the XML file.		
Prerequisites			
Syntax	bool WriteData(TCHAR *data, int indent = 0);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
data	In	TCHAR	string data
indent	In	Int	Number of tabs (tab = 4 spaces) to indent the host tag data.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

WriteIndent()			
Description	Writes 4 spaces to the XML file.		
Prerequisites			
Syntax	void WriteIndent(int indent);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
indent	In	Int	Number of idents.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

C++ and Java Code Examples

```
//set the xml file path and name
xmllaunch.XMLCreate((char*)strXMLFullFileName.c_str());

if (FAILED(m_pIEAGameControlHost->GetGameControlHostVersion(&lBSTR)))
       return false;
bstr = lBSTR;
str = bstr;

VariantClear(&var);

xmllaunch.MMLaunchCreate(TAG_FILEID, "MMLAUNCH SYSTEM \"MMLaunch.dtd\"",
                        (char*)str.c_str(), "1.2");

if (FAILED(m_pIEAGameControlHost->GetGameControlHostPlayerData (OwnerBLO-
BId,PROPERTY_IP,&var)) || var.vt != VT_BSTR)
       return false;
bstr = var.bstrVal;
str = bstr;

VariantClear(&var);

//write out the host info...
if (PBLOBId == OwnerBLOBId)
       xmllaunch.HostInfo(ATTR_CLIENTTYPE, ATTR_CLIENTTYPE_LOCAL,
       (char*)strOwner.c_str(), (char*)str.c_str(), 1);
else
       xmllaunch.HostInfo(ATTR_CLIENTTYPE, ATTR_CLIENTTYPE_REMOTE,
       (char*)strOwner.c_str(), (char*)str.c_str(), 1);

//get this players IP adress
```

```
if (FAILED(m_pIEAGameControlHost->GetGameControlHostPlayerData(lVal,PROPERTY_IP,&var)) ||
         var.vt != VT_BSTR)
        return false;
bstr = var.bstrVal;
str = bstr;

VariantClear(&var);

//is this a local or remote client
if (PBLOBId == lVal)
        xmllaunch.ClientInfo("clientType", "local", (char*)strPlayer.c_str(),
        (char*)str.c_str(), 1);
else
        xmllaunch.ClientInfo("clientType", "remote", (char*)strPlayer.c_str(),
        (char*)str.c_str(), 1);

xmllaunch.WriteData("<!-- ranked game -->\n", 0);
xmllaunch.GameSettings(1, 0, 1);
xmllaunch.WriteData("<!-- game type -->\n", 0);
xmllaunch.GameSettings(2, GS.GetGameSetting(GS.GameTypeDataID), 1);
xmllaunch.WriteData("<!-- offensive fouls -->\n", 0);
xmllaunch.GameSettings(3, GS.GetGameSetting(GS.OffensiveFoulsDataID), 1);
xmllaunch.WriteData("<!-- defensive fouls -->\n", 0);
xmllaunch.GameSettings(4, GS.GetGameSetting(GS.DefensiveFoulsDataID), 1);

_stprintf(tBuf, _T("%ld"),PS.GetTeam());
xmllaunch.PlayerSettings((TCHAR*)strPlayer.c_str(), _T("1"), (TCHAR*)tBuf, 1);

// add date
char tBuf[256] = {NULL};
sprintf(tBuf, "%.4d%.2d%.2d", GS.m_sysTime.wYear, GS.m_sysTime.wMonth,
        GS.m_sysTime.wDay);
xmllaunch.WriteTaggedData("HOST_DATE", tBuf, 1);

xmllaunch.WriteTaggedData("CLIENT_COUNT", NumPlayers, 1);

xmllaunch.XMLWriteComplete(TAG_FILEID);
```

XML File Code Sample

```
xmllaunch.WriteTaggedData("CLIENT_COUNT", NumPlayers, 1);

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE MMLAUNCH SYSTEM "MMLaunch.dtd">
<MMLAUNCH><MM_VERSION>1,0,0,2</MM_VERSION>
<GAME_VERSION>1.2</GAME_VERSION>
    <HOST clientType="local">
        <NAME>echo</NAME>
        <IPADDRESS>10.14.142.162</IPADDRESS>
    </HOST>
    <CLIENT clientType="remote">
        <NAME>kim</NAME>
        <IPADDRESS>10.14.142.15</IPADDRESS>
    </CLIENT>
<!-- ranked game -->
    <GAME_SETTING>
```

```
        <GAME_SETTING_ID>1</GAME_SETTING_ID>
        <GAME_SETTING_VALUE>0</GAME_SETTING_VALUE>
    </GAME_SETTING>
<!-- game type -->
    <GAME_SETTING>
        <GAME_SETTING_ID>2</GAME_SETTING_ID>
        <GAME_SETTING_VALUE>0</GAME_SETTING_VALUE>
    </GAME_SETTING>
<!-- offensive fouls -->
    <GAME_SETTING>
        <GAME_SETTING_ID>3</GAME_SETTING_ID>
        <GAME_SETTING_VALUE>5</GAME_SETTING_VALUE>
    </GAME_SETTING>
<!-- defensive fouls -->
    <GAME_SETTING>
        <GAME_SETTING_ID>4</GAME_SETTING_ID>
        <GAME_SETTING_VALUE>5</GAME_SETTING_VALUE>
    </GAME_SETTING>


<!-- team -->
    <PLAYER_SETTING>
        <NAME>echo</NAME>
        <PLAYER_SETTING_ID>1</PLAYER_SETTING_ID>
        <PLAYER_SETTING_VALUE>1369</PLAYER_SETTING_VALUE>
    </PLAYER_SETTING>


    <HOST_DATE>524285242852428</HOST_DATE>
    <CLIENT_COUNT>2</CLIENT_COUNT>
</MMLAUNCH>
```

EA Game Control Interface Description

Interface (IEAGameControl)

The IEAGameControl interface is used for communicating from a game control host to a game control.

AllowReady()			
Description	This interface method is called when a player clicks ready. If the player should not be allowed to begin game play, the player is prohibited from readying himself or herself. If the player is not allowed to enter, LastError should explain the reason.		
Prerequisites			
Syntax	HRESULT AllowReady([in] long GameBLOBID, [in] long PlayerBLOBID, [out] BOOL *pAllowAction)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
GameBLOBID	In	Long	An identifier that can be passed to the host to obtain the BLOB (Binary Larger Object).
PlayerBLOBID	In	Long	An identifier that can be passed to the host to obtain the BLOB for the selected player's individual settings
pAllowAction	Out	BOOL	Assigned a return value that specifies whether the action (readying) should be allowed.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

AllowRoomEntry()			
Description	This interface method is the same as AllowReady, but is called when a player would like to enter a room.		
Prerequisites			
Syntax	HRESULT AllowRoomEntry([in] long GameBLOBID, [in] long PlayerBLOBID, [out] BOOL *pAllowAction)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
GameBLOBID	In	Long	An identifier that can be passed to the host to obtain the BLOB (Binary Larger Object).
PlayerBLOBID	In	Long	A long identifier that can be passed to the host to obtain the BLOB for the selected player's individual settings.
pAllowAction	Out	BOOL	Assigned a return value that specifies whether the action (room entry) should be allowed.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetDefaultGameSettings()			
Description	This interface method is called to retrieve the initial BLOB for each game. It is also called if the room owner clicks the “default” button after making changes.		
Prerequisites			
Syntax	HRESULT GetDefaultGameSettings([in, out] VARIANT *pGameSettingsDataArray)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
pGameSettingsDataArray	In/Out	VARIANT	A pointer to a VARIANT that the calling process must have set to type VT_ARRAY|VT_UI1 or VT_EMPTY (in which case the game control will convert it to type(ARRAY|VT_UI1). The game settings should be packed into this VARIANT as a SafeArray of type VT_ARRAY|VT_UI1. Also, the array size should be minimized as this is marshalled over the Internet.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetDefaultPlayerSettings()			
Description	This interface method is the same as GetDefaultGameSettings except that it is used for retrieving default settings for each player.		
Prerequisites			
Syntax	HRESULT GetDefaultPlayerSettings([in, out] VARIANT *pPlayerSettingsDataArray)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
pPlayerSettingsDataArray	In/Out	VARIANT	A VARIANT that the calling process must have set as type VT_ARRAY|VT_UI1 or VT_EMPTY. The game settings should be packed into this VARIANT as a SafeArray of type VT_ARRAY|VT_UI1. Also, the array size should be minimized as this is marshalled over the Internet.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetModifiedGameSettings()			
Description	This interface method is called to retrieve the new game BLOB when apply is clicked (and game settings have changed). The same rules and parameters apply to it as GetDefaultGameSettings. Currently displayed information should be packed.		
Prerequisites			
Syntax	HRESULT GetModifiedGameSettings([in, out] VARIANT *pGameSettingsDataArray)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
pGameSettingsDataArray	In/Out	VARIANT	A VARIANT that the calling process must have set as type VT_ARRAY|VT_UI1 or VT_EMPTY. The game settings should be packed into this VARIANT as a SafeArray of type VT_ARRAY|VT_UI1. Also, the array size should be minimized as this is marshalled across the Internet.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetModifiedPlayerSettings()			
Description	This interface method is the same as GetModifiedGameSettings except for player settings. The same rules and parameters apply to it as GetDefaultPlayerSettings.		
Prerequisites			
Syntax	HRESULT GetModifiedPlayerSettings([in, out] VARIANT *pPlayerSettingsDataArray)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
pPlayerSettingsDataArray	In/Out	VARIANT	
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetPlayerListColumnData()			
Description	This interface method is called to retrieve display information for a player's settings for a specific player BLOB.		
Prerequisites			
Syntax	HRESULT GetPlayerListColumnData([in] long PlayerBLOBID, [in] long DataID, [out] VARIANT *pItemValue)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
PlayerBLOBID	In	Long	The BLOB ID for the player settings for which the host wants display information.
DataID	In	Long	Game control-specified identifier of a display column which was previously registered by the game control.
pItemValue	Out	VARIANT	pItemValue is a pointer to a VARIANT which is set to the display value (and type).
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetGameListColumnData()

Description	This interface method is called to retrieve display information for a game setting for a specific game BLOB.
Prerequisites	
Syntax	HRESULT GetGameListColumnData([in] long GameBLOBID, [in] long DataID, [out] VARIANT *pItemValue)

Arguments	**In, Out, In/Out**	**Type**	**Description**
GameBLOBID	In	Long	The BLOB ID for the game settings for which the host wants display information.
DataID	In	Long	This is the ID of the display column previously registered by the game control specifying what data to return.
pItemValue	Out	VARIANT	pItemValue is a pointer to a variant that retrieves the display value (and type).

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

LaunchGame()

Description	LaunchGame is called to launch the game. This interface method is highly title specific. The parameter for the game must be packed in a manner that the game is expecting and the game requires so that it can launch.
Prerequisites	
Syntax	HRESULT LaunchGame([in] long GameBLOBID, [in] long LocalPlayerIndex, [in] VARIANT PlayerBLOBIDArray, [in] NumPlayers, [out] unsigned long *waitHandle)

Arguments	In, Out, In/Out	Type	Description
GameBLOBID	In	Long	This is a BLOB ID for the game being launched.
LocalPlayerIndex	In	Long	The index into PlayerBLOBIDArray that holds our local player's BLOBID.
PlayerBLOBIDArray	In	VARIANT	PlayerBLOBIDArray is a safe array of type VT_UI4 that specifies the player BLOB IDs for each player in the game.
NumPlayers	In	Long	Number of players playing and size of PlayerBLOBIDArray.
waitHandle	Out	Long	waitHandle must be set to some waitable window handle (such as an Event or anything that win32 WaitForSingleEvent will block on) that will be signaled when the game is done (or aborted).

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes	If the selected game is a browser game, the IGameControlHost method "Navigate" should be called to open a new browser window (the JavaScript method 'navigate ()' does not pop up a new window automatically) from the current browser so the server session information is kept and auth/auth tickets are still valid.	

Setup()

Description	Setup is an initialization method called by the Matchmaker for the initial setup.
Prerequisites	
Syntax	HRESULT Setup([in] VARIANT pGameControlHostUnknown, [in] VARIANT pPropertyBagUnknown, [in] BSTR language)

Arguments	In, Out, In/Out	Type	Description
pGameControlHostUnknown	In	VARIANT	pGameControlHostUnknown is a VARIANT of type VT_UNKNOWN holding the IUnknown of the IGameControlHost.
pPropertyBagUnknown	In	VARIANT	pPropertyBagUnknown is a VARIANT of type VT_UNKNOWN holding the IUnknown of an IpropertyBag, which contain the parameter tags from the Web page.
language	In	BSTR	Language is an ISO 639 string that specifies the language to use.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

ShowView()

Description	ShowView is a method called by the host to change or refresh the active view.
Prerequisites	
Syntax	HRESULT ShowView([in] long GameBLOBID, [in] long ViewID, [in] BOOL bModifiable, [in] BOOL PlayerBLOBID, [in] BOOL bIsThisInitialSetup)

Arguments	In, Out, In/Out	Type	Description
GameBLOBID	In	Long	A Game BLOB identifier that corresponds to what information should be shown.
ViewID	In	Long	ViewID is an identifier known by the game control to specify what settings view to show.
bModifiable	In	BOOL	bModifiable is a pointer to a BOOL which should be set or unset to specify whether the current user has the ability to modify settings for the current view.
PlayerBLOBID	In	BOOL	Player BLOB that corresponds to what should be shown.
bIsThisInitialSetup	In	BOOL	bIsThisInitialSetup specifies whether this is the create game step. It permits the view to customize itself for initial setup.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

Look and Feel and Error Properties Method

The following method is used to get the game control's preferred look and feel settings. The exception is LastError, which should have a description of any current error.

GetGameControlSetting ()			
Description	Method that host reqest to retrieve look and feel settings for the game control.		
Prerequisites			
Syntax	HRESULT GetGameControlSetting(long lDataID, VARIANT *pOutVar)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
lDataID	In	Long	This is the known numeric value telling the game control what information it is requesting. The values and meanings are defined in mmconstants.h (e.g.; EAGC_LONG_MMSKIN_PATH, EAGC_OLECOLOR_LIST_BACK)
pOutVar	In	VARIANT	Pointer to a variant to fill with the information requested.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

Unsetup()

Description	As part of its cleanup, the host calls this method. Any cleanup can be done here, but more importantly the host is expecting the control to release any references it has on the IGameControlHost.		
Prerequisites			
Syntax	HRESULT Unsetup()		
Arguments	**In, Out, In/Out**	**Type**	**Description**
None			
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

HostSettingsChanged ()

Description	The HostSettingsChanged interface method is called by the host to inform the game control that hosted settings have changed
Prerequisites	
Syntax	HRESULT HostSettingsChanged([in] long GameBLOBID, [out] BOOL *pGameSettingsChanged, [in] long PlayerBLOBID, [out] BOOL *pPlayerSettingsChanged)

Arguments	In, Out, In/Out	Type	Description
GameBLOBID	In	Long	Applicable game's BLOB ID.
pGameSettingsChanged	Out	BOOL	An out value that the game control uses to inform the host that game settings were modified in order to be in compliance with the host settings.
PlayerBLOBID	In	Long	Applicable Player's BLOB ID.
pPlayerSettingsChanged	Out	BOOL	An out value that the game control uses to inform the host that player settings were modified in order to be in compliance with the host settings.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

AddLobbyGameListColumn()

Description	AddLobbyGameListColumn is used to add a column to the list display of games in the lobby.
Prerequisites	
Syntax	HRESULT AddLobbyGameListColumn([in] long Width, [in] BSTR Heading, [in] BYTE DataSource, [in] long DataID)

Arguments	In, Out, In/Out	Type	Description
Width	In	Long	Use this parameter to define the number of pixels wide a column should be for this column (approx. 30-50 for numbers and 75-125 for text.)
Heading	In	BSTR	Specifies the column heading.
DataSource	In	BYTE	The DataSource specified can be either Matchmaker or game control. Use MACROS in MMConstants.h DATASOURCE_Matchmaker and DATASOURCE_GAMECONTROL)
DataID	In	Long	For DATASOURCE DATASOURCE_Matchmaker use MACROS in MMConstants.h. If DataSource is DATASOURCE_GAMECONTROL, this value is specified by the game control and will be passed back to retrieve information for the current column.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

AddLobbyPlayerListColumn()

Description	AddLobbyPlayerListColumn is used to add a column to the list display of players in the lobby.
Prerequisites	
Syntax	HRESULT AddLobbyPlayerListColumn([in] long Width, [in] BSTR Heading, [in] BYTE DataSource, [in] long PlayerBLOBDataID)

Arguments	In, Out, In/Out	Type	Description
Width	In	Long	Use this parameter to define the number of pixels wide a column should be (approx. 30-50 for numbers and 75-125 for text.)
Heading	In	BSTR	Specifies column heading.
DataSource	In	BYTE	The DataSource specified can be either Matchmaker or game control. Use MACROS in MMConstants.h (DATASOURCE_Matchmaker and DATASOURCE_GAMECONTROL)
PlayerBLOBDataID	In	Long	For DATASOURCE DATASOURCE_Matchmaker use MACROS in MMConstants.h. If DataSource is DATASOURCE_GAMECONTROL, this value is specified by the game control and will be passed back to retrieve information for the current column.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

AddRoomPlayerListColumn()

Description	AddRoomPlayerListColumn is used to add a column to the list display of players in the game room.
Prerequisites	
Syntax	HRESULT AddRoomPlayerListColumn([in] long Width, [in] BSTR Heading, [in] BYTE DataSource, [in] long PlayerBLOBDataID)

Arguments	In, Out, In/Out	Type	Description
Width	In	Long	Use this parameter to define the number of pixels wide a column should be (approx. 30-50 for numbers and 75-125 for text.)
Heading	In	BSTR	Specifies column heading.
DataSource	In	BYTE	The DataSource specified can be either Matchmaker or game control. Use MACROS in MMConstants.h (DATASOURCE_Matchmaker and DATASOURCE_GAMECONTROL) If DataSource is DATASOURCE_GAMECONTROL, this value is specified by the game control and will be passed back to retrieve information for the current column.
PlayerBLOBDataID	In	Long	If DataSource is DATASOURCE_GAMECONTROL, this value will be passed back to retrieve information for current column.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

AddView()

Description	AddView is used to register a view for a player or game settings.
Prerequisites	
Syntax	HRESULT AddView([in] BSTR Heading,[in] long ViewID, [in] BOOL IsPlayerView)

Arguments	In, Out, In/Out	Type	Description
Heading	In	BSTR	This string is used to specify the heading for whatever object is used to select the current view (probably a tab).
ViewID	In	Long	A View identifier that the game control uses to determine which interface to display.
IsPlayerView	In	BOOL	IsPlayerView specifies to the host whether this is a player view (or by default a game settings)

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

GetGameSettings()			
Description	This interface method retrieves a BLOB from the host.		
Prerequisites			
Syntax	HRESULT GetGameSettings([in] long GameBLOBID, [in,out] VARIANT *pGameSettingsDataArray)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
GameBLOBID	In	Long	GameBLOBID is the identifier of the game BLOB you are trying to retrieve.
pGameSettingsDataArray	In	VARIANT	pGameSettingsDataArray is a VARIANT of type VT_EMPTY or VT_ARRAY|VT_UI1 that you would like filled with the game settings BLOB.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetGameControlHostVersion()			
Description	GetGameControlHostVersion is used to retrieve the game control's host version number.		
Prerequisites			
Syntax	HRESULT GetGameControlHostVersion([out] BSTR *pHostVersion)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
pHostVersion	Out	BSTR	pHostVersion pHostVersion is a pointer to a BSTR string you would like filled with the host's version number.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetPlayerSettings()			
Description	This method is used to retrieve a BLOB from the host.		
Prerequisites			
Syntax	HRESULT GetPlayerSettings([in] long PlayerBLOBID, [in,out] VARIANT *pPlayerSettingsDataArray)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
PlayerBLOBID	In	Long	PlayerBLOBID is an identifier of the player BLOB you are trying to retrieve.
pGameSettingsDataArray	In/Out	VARIANT	pGameSettingsDataArray is a VARIANT of type VT_EMPTY or VT_ARRAY|VT_UI1 that you would like filled with the BLOB. May result in a call back to the Game Control.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GameSettingsChanged()			
Description	GameSettingsChanged informs the host that game settings have changed.		
Prerequisites			
Syntax	HRESULT GameSettingsChanged([in] long GameBLOBID, [in] long ViewID, [in] BOOL bAutoApply)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
GameBLOBID	In	Long	BLOB ID in which settings have changed (if they are retrieved).
ViewID	In	Long	The view ID the game settings have changed in.
bAutoApply	In	BOOL	bAutoApply is used to specify whether to enable the apply button, or just have the host retrieve the new game BLOB from the control automatically.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

PlayerSettingsChanged()

Description	PlayerSettingsChanged informs the host that player settings have changed.
Prerequisites	
Syntax	HRESULT PlayerSettingsChanged([in] long PlayerBLOBID, [in] long ViewID, [in] BOOL bAutoApply)

Arguments	In, Out, In/Out	Type	Description
PlayerBLOBID	In	Long	BLOB ID in which settings have changed (if they are retrieved).
ViewID	In	Long	The view ID that the game settings have changed in.
bAutoApply	In	BOOL	bAutoApply is used to specify whether to enable the apply button, or just have the host retrieve the new BLOB automatically.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

GetGameControlHostPlayerData ()

Description	GetGameControlHostPlayerData is used to retrieve a setting from the host.
Prerequisites	
Syntax	HRESULT GetGameControlHostPlayerData([in] long PlayerBLOBID, [in] long DataID, [in, out] VARIANT *pItemValue)

Arguments	In, Out, In/Out	Type	Description
PlayerBLOBID	In	Long	The BLOB ID for the current player (used to identify the player).
DataID	In	Long	DataID is the constant identifier for the desired information. These are specified in MMConstants.h (PROPERTY_ROOMID, PROPERTY_PLAYERID,…).
pItemValue	In/Out	VARIANT	pItemValue is a pointer VARIANT you would like filled with the information.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

GetGameControlHostGameData()

Description	GetGameControlHostGameData is used to retrieve a setting from the host.
Prerequisites	
Syntax	HRESULT GetGameControlHostGameData([in] long GameBLOBID, [in] long DataID, [in, out] VARIANT *pItemValue)

Arguments	**In, Out, In/Out**	**Type**	**Description**
GameBLOBID	In	Long	The BLOB ID for the current game (used to identify the game)
DataID	In	Long	DataID is the constant identifier for information wanted by the control. This information is listed in MMConstants.h (PROPERTY_ROOMID, PROPERTY_OWNER).
pItemValue	In/Out	VARIANT	pItemValue is a VARIANT you would like filled.

Error Messages		
Return Values		0 = True on success. 1 = HRESULT error code on failure.
Exceptions		
Notes		

GetServiceID()			
Description	GetServiceID is used to retrieve the service ID from the host (not yet implemented as it is not yet defined).		
Prerequisites			
Syntax	HRESULT GetServiceID([out] BSTR *pServiceID)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
pServiceID	Out	BSTR	pServiceID is a string you would like filled with the current game's service ID.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

GetGameTechID()			
Description	GetGameTechID is used to retrieve the game tech ID from the host (not yet implemented as it is not yet defined).		
Prerequisites			
Syntax	HRESULT GetGameTechID([out] BSTR *pGameTechID)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
pGameTechID	Out	BSTR	pGameTechID is a string you would like filled with the current game's Game Tech ID.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

Navigate ()			
Description	Navigate is used to cause some desired activity in the browser in which the Matchmaker is ultimately embedded. If the Matchmaker is not embedded in a browser that supports this type of functionality it does nothing.		
Prerequisites			
Syntax	HRESULT Navigate([in] BSTR url)		
Arguments	**In, Out, In/Out**	**Type**	**Description**
URL	In	BSTR	The URL you would like the host to navigate to.
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes	If the host is embedded in the page, this will cause it to be unloaded. This will usually be used to call Javascript on the page (e.g., “javascript: myembeddedmethod(‘my param’);”).		

MMLaunchCreate()			
Description	Writes the XML file header description tags and info.		
Prerequisites			
Syntax	bool MMLaunchCreate(TCHAR *xmlName, TCHAR *docType, TCHAR *mmVersion, TCHAR *gameVersion);		
Arguments	**In, Out, In/Out**	**Type**	**Description**
xmlName	In	TCHAR	“TAG_FILEID” which is “MMLAUNCH”.
docType	In	TCHAR	"MMLAUNCH SYSTEM \"MMLaunch.dtd\".
mmVersion	In	TCHAR	Matchmaker version (The string value obtained from the Matchmaker API call GetGameControlHostVersion).
gameVersion	In	TCHAR	Game version (hard coded to 1.2).
Error Messages			
Return Values		0 = True on success. 1 = HRESULT error code on failure.	
Exceptions			
Notes			

Definitions

BLOB

Each set of game data (e.g.; player settings, game setting, Matchmaker room settings) are passed as a BLOB (Binary Larger Object).

Return type

All methods return HRESULTs

Technical Requirements – Localized Matchmaker Capability

Matchmaker Client v.2

Requirement ID	Technical Requirement	Description
1.1.1	UTF-8 support	Data Storage Conversion Providing Support for ANSI and Unicode. Functions Conversion to Support ANSI and Unicode.
1.1.2	LANGUAGE_ID	Handle Language Code (passed from ESD or GHPs)
1.1.3	Resource DLLs	Move strings to Language Specific Resource DLL
1.1.4	Matchmaker Client v.2	The Matchmaker client will be physically stored on the user's machine in a stand-alone "Matchmaker" directory. The Matchmaker client will load language-specific resource DLL from the Language Code it receives. In the absence of a language code, the Matchmaker client should default to the English resource DLL. The Matchmaker client default UI has been modified to display localized text and art across all supported languages.
1.1.5	Matchmaker Game Control	Each game control will handle the Language Code (passed from the Matchmaker client). The game control will be physically stored on the user's machine in each game's directory. The Matchmaker game control is installed with the game and is localized for the installed version of the game.

Table of Contents

Component History 1
Document History 1
Introduction to Matchmaker 1.0 2
Features 2
Developer Needs 2
Frequently Asked Questions 3
Matchmaker Game Control – Getting Started 6
Steps to Create a Game Control 6
DlgDrawer API 10
Files Included in the DlgDrawerAPI ZIPfile 10
DlgDrawer Overview 10
DlgDrawer Programming Guide 10
KickApp API 12
Files Included in the Kick Application API ZIPfile 12
KickApp API Overview 12
KickApp Client Interfaces 12
XML Write API 14
Files Included in the XML WriteAPI ZIPfile 14
XML Write API Overview 14
XML Client Interfaces 14
EA Game Control Interface Description 28
Technical Requirements – Localized Matchmaker Capability 55