FantastiqUI Documentation Version 1.0

About

FantastiqUI allows you to use flash movies (.SWF) in hardware accelerated environments on both 2d and 3d surfaces. Using a powerful C++ core the Flash player is projected onto a texture surface, after which the texture can be used as you would normally use textures. Mouse and keyboard input is fully handled by **FantastiqUI** allowing flash movies to be used for interface purposes as well. Support for transparency as well as translucency allows you to make fantastic user interfaces in the Flash authoring environment in little time. C++ as well as the higher level .NET languages are supported by **FantastiqUI**

About documentation

This documentation is mostly a function reference. Where more details are required, small code examples will be given in the function reference.

Class overview

FUIMain (.NET : **FNUIMain** COM : **ICFUIMain**)

About this class:

This is the main GUI class. This class contains one GUI processing thread and all child flash players are processed in this thread. It is allowed to have multiple active FUIMain classes to use multiple threads.

Functions:

```
int CreateUI(void* pResizeTexCallback,void* pGetSurfCallback,
void* pReleaseSurfCallback,char* sFlashControl,void* pDirtyRect);
```

Variable	Description
pResizeTexCallback	Pointer(Delegate in .NET) for the resize texture callback
pGetSurfCallback	Pointer(Delegate in .NET) for the resize texture callback
pReleaseSurfCallback	Pointer(Delegate in .NET) for the resize texture callback
sFlashControl	Path of the flash control ocx, NULL for default system flash control ocx.
pDirtyRect	Pointer(Delegate in .NET) for the resize texture callback

This functions initializes the class.

```
void SetFlashPath(char* fpath);
```

Set the path of the flash player .ocx to use.

```
FUIFlashPlayer* CreateFlashPlayer();
```

Creates a flash player class.

```
void DeleteFlashPlayer(FUIFlashPlayer* pPlayer);
```

Deletes a flash player class.

```
int LoadFlashHeader(char* file,int* x,int* y,float* fps,int* numframes);
```

Loads the specifed flash movie, and return its size, framerate and the number of frames in the movie. Useful to create a flash player that exactly matches the size of the movie loaded.

Callbacks:

```
void* __stdcall _ResizeTexture(void* p,void* _pTexture,int iSizeX,int
iSizeY,int iReserved)
```

Variable	Description
p	Unique identifier of the flash player class involved. See pUnique in CreateFlashControl()
pTexture	Unique identifier of the texture involved.
iSizeX	Size of the new texture
iSizeY	Size of the new texture
iReserved	Reserved

This callback is called when the flash textures need to be resized. As returnvalue a new texture identifier can be passed back to FantastiqUI. After returning the texture will then be identified using the new texture identifier.

Warning : This function is called from the flash thread, therefore it is important to be careful with multithreading issues in the implementation of this callback.

```
void* __stdcall _GetTextureSurfacePointer(void* p,void* pTexture)
```

Variable	Description
p	Unique identifier of the flash player class involved. See pUnique in CreateFlashControl()
pTexture	Unique identifier of the texture involved.

This callback is called when FantastiqUI needs to output the flash player pixel data. The return value is a pointer to a surface where FantastiqUI can safely write the pixel data to. For more information about texture sizes, see CreateFlashControl().

Warning : This function is called from the flash thread, therefore it is important to be careful with multithreading issues in the implementation of this callback.

```
int __stdcall _ReleaseTextureSurfacePointer(void* p,void* pTexture,void*
pPointer)
```

Variable	Description
p	Unique identifier of the flash player class involved. See pUnique in CreateFlashControl()
pTexture	Unique identifier of the texture involved.

pPointer	Pointer that was given to FantastiqUI as surface pointer

This callback is called when FantastiqUI has finished writing to the surface. From this point FantastiqUI will no longer access the surface and normally this is the location to upload the surface to video memory.

Warning : This function is called from the flash thread, therefore it is important to be careful with multithreading issues in the implementation of this callback.

```
void __stdcall _DirtyRect(void* p,void* pTexture,int x,int y, int x1,int y1)
```

Variable	Description
P	Unique identifier of the flash player class involved. See pUnique in CreateFlashControl()
pTexture	Unique identifier of the texture involved.
X, Y	Top left corner of dirty rectangle
X2, Y2	Bottom right corner of dirty rectangle

This callback is called to inform the user of a dirty region on the texture surface. This callback is useful because some rendering API's can upload the texture to the video card faster when dirty region's are specified to the API.

Warning : This function is called from the flash thread, therefore it is important to be careful with multithreading issues in the implementation of this callback.

FUIFlashPlayer (.NET : **FNUIFlashPlayer** COM : **ICFUIFlashPlayer**)

About this class:

This is the flash player class. It loads and plays .swf movies and renders them to any memory surface (texture) specified.

Creation/Loading movies:

```
int CreateFlashControl(int iTransparency,int iSizeX,int iSizeY,void*
pTexture1=NULL,void* pTexture2=NULL,bool b2NTextures=false,void*
pUniqueId=NULL);
```

Variable	Description
iTransparency	Transparency mode 0=No transparency 1=Basic transparency 2=Full transparency
iSizeX	Horizontal player size
iSizeY	Vertical player size
pTexture1	First texture
pTexture2	Second texture
b2NTextures	Assume textures are 2^n?
pUniqueId	Unique ID specifying which flash player is involved in the callback functions.

iTransparency specifies the transparency mode to use. No transparency is the fastest option, to be used when the movie rendered does not require any transparency/translucency. Basic transparency provides

correct transparency in most cases, but due to bugs in the Adobe Flash Player in some cases you may see transparency error's. In this case, use Full Transparency.

pTexture1 and pTexture2 provides the texture identifiers of the textures to draw flash to. When only pTexture1 is passed single buffering will be used, When both texture identifiers are passed multitexturing will be used. These texture identifiers are the identifiers passed to the callback functions.

When b2NTextures is set to true, FantastiqUI will assume the textures flash is drawn to are rounded up to a 2^n size. Older video cards require textures to be 2^n.

```
int LoadMovie(wchar_t* sMovie);
```

Load the specified flash movie.

```
int Resize(int iNewSizeX,int iNewSizeY);
```

Resize the flash player control. Calling this function will invoke the ResizeTexture() callback/delegate.

Input:

```
void UpdateMouseButton(int iButton,bool bPressed);
```

Sends the state of the mouse buttons to FantastiqUI player control. This function only has to be called when the mouse button state changes, but calling it every frame incurs no performance penalty.

```
void UpdateMousePosition(int iX, int iY);
```

Sends the position of the mouse cursor to this FantastiqUI player control.

```
void DoCopy();
```

Invokes the equivalent of Ctrl+C, normally you will want to link this to the Ctrl+V key combination or a menu option. When calling this function for a key combination, do not call SendKey() or SendChar() for these keys.

```
void DoPaste();
```

Invokes the equivalent of Ctrl+C, normally you will want to link this to the Ctrl+V key combination or a menu option. When calling this function for a key combination, do not call SendKey() or SendChar() for these keys.

```
void DoSelectAll();
```

Invokes the equivalent of Ctrl+C, normally you will want to link this to the Ctrl+V key combination or a menu option. When calling this function for a key combination, do not call SendKey() or SendChar() for these keys.

```
void DoCut();
```

Invokes the equivalent of Ctrl+C, normally you will want to link this to the Ctrl+V key combination or a menu option. When calling this function for a key combination, do not call SendKey() or SendChar() for these keys.

```
void DoMouseWheel(int iScroll);
```

Sends the state of the mouse wheel to flash, only call this function when the mouse wheel was turned.

```
void SendKey(bool bDown,int iVirtualKey,int iExtended);
```

Variable	Description
bDown	State of the key specified
iVirtualKey	Keycode of the key pressed.
iExtended	Same as iExtended in WM_KEYDOWN

Sends keyboard input to this FantastiqUI flash player. Equivalent of WM_KEYDOWN/WM_KEYUP windows message. Only call this function when a key's state has changed.

```
void SendChar(int iChar,int iExtended);
```

Variable	Description
iChar	Character code of the character
iExtended	Same as iExtended in WM_CHAR

Sends character input to this FantastiqUI flash player. Equivalent of WM_CHAR windows message.

Texturing:

The texturing functions are useful especially in the case of double buffering. The GetTexture() will return the texture identifier of the texture to be used for rendering.

Warning : GetTexture() will also internally lock the texture returned for updating. This is important, without the lock FantastiqUI might attempt to update the texture contents while the user is using the texture for rendering purposes. ReleaseTexture() releases this internal lock again.

```
void* GetTexture();
```

Returns the texture identifier of the texture to use and locks it internally for updating

```
void ReleaseTexture();
```

Releases the internal lock on the texture retrieved using GetTexture()

Calling actionscript functions:

```
void PrepareFunctionCall(wchar_t *funcname);
```

To call a function inside the actionscript code of the loaded movie, first call this function to specify the name of the function to call.

After calling this function, all arguments for the function can be provided using the PushArgument() functions.

```
void PushStringArgument(wchar_t *arg);
```

Push a string argument.

```
void PushNumberArgument(float arg);
```

Push a number(float) argument.

```
void PushBoolArgument(bool arg);
```

Push a boolean argument.

```
bool FinishCall(bool bReturnArgument);
```

When finished pushing all the arguments for the function call, call this function to execute the call. bReturnArguments specifies whether the user immeadiately requires return arguments. Setting this variable to true means you can immediately get the return arguments from the event list after this call completes. This is slower than setting bReturnArguments to false, since the function can not return untill ActionScript is done processing.

Returning arguments to flash when flash calls the host langauge:

When ActionScript makes a call to the host language, the EventNotifier() callback/delegate is invoked. Inside this callback, you can return arguments to ActionScript using the functions specified below.

```
void PrepareReturn();
```

Call this function to prepare for returning arguments to ActionScript. After calling this function you can provide all arguments to return using the PushArgument() functions specified earlier.

```
void ReturnArguments();
```

This function executes the return of the arguments specified to ActionScript.

Events from flash:

FantastiqUI stores all events the flash player raises in an event list. The functions specified below can be used to access this event list. For more information about events see the Event class.

```
int GetNumEvents();
```

Get the number of events in the event list.

Warning : This function also locks the event list internally. Therefore it is important to call ClearEvents() as soon as you are finished accessing the event list, since this function unlocks the event list internally.

```
FUIFlashEvent* GetEvent(int iNum);
```

Get the event specified by iNum.

```
void DeleteEvent(FUIFlashEvent* p);
```

Use this function to erase the memory allocated by an event class retrieved using GetEvent().

```
void ClearEvents();
```

Clear the full event list. This function also unlocks the event list internally.

```
void SetEventNotifier(void* notifier);
```

Set the event notifier callback/delegate.

Flash settings:

```
void SetFlashSettings(bool localsecurity,int allownetworking,bool
allowscriptaccess,int backgroundcolor);
```

Variable	Description
Localsecurity	True = Enforce local security False = Disable local security
allownetworking	0 = “All” 1 = “Internal” 2 = “None”
allowscriptaccess	True = “always” False = “never”
backgroundcolor	Background color of the flash movie, only used when transparency mode is set to Opaque.

```
void SetFrameTime(float frametime);
```

Set the time per frame in milliseconds at which the flash movie is rendered. Frametime = 1000.0 / framerate.

FUIFlashEvent(.NET : **FNUIFlashEvent** COM : **ICFUIFlashEvent**)

About this class:

This class contains events received from Flash. This class is obtained from the FUIFlashPlayer class by calling GetEvent(i). Multiple types of events exist, the type of the current event can be retrieved using GetEventType().

Functions:

```
int GetEventType();
```

Get the type of the current event.

```
wchar_t* GetFSCommand();
```

Get the name of the function specified in the FSCommand() call in ActionScript. Only use this function when the event type is FSCommand.

```
wchar_t* GetFunctionName();
```

Get the function name specified in the ExternalObjects call in ActionScript.

```
int GetNumArguments();
```

Get the number of arguments specified with the call in ActionScript.

```
int GetValueType(int v);
```

Get the type of argument v.

Use the following functions to get the actual argument values for argument v.

- `wchar_t* GetValueString(int v);`
- `int GetValueNumber(int v);`
- `float GetValueFloat(int v);`
- `bool GetValueBool(int v);`

Appendix

Setting up a function in ActionScript that can be called from the host language:

First, create the actual function :

```
//callback function
function CallbackTest(arg1) : String{
      text2.text = arg1;
      return(text1.text);
}
```

After that, register the function using the ExternalInterface class, the first parameter is the name assigned to the function when called from the host language. The second is the actual function in ActionScript

```
//register the callback function
var connection = ExternalInterface.addCallback("CallbackTest", CallbackTest);
```

Calling a function in the host language:

```
ExternalInterface.call("sayHello", arguments);
```