Multimedia Fusion iOS extension SDK Version July 2011

Welcome to the MMF2 iOS extension SDK. Please read this file up to the end. After reading it, you will be able to create new extensions for the MMF2 iOS runtime. If you are fluent with the extension kit on other langages (like Java or Flash), you will see that the iOS extension kit is very similar.

This extension kit contains all the source code of the iOS runtime. I (Francois) would be really happy to hear any suggestion you have about my source code, as it might help me improve it. If you have an idea about the source code, please write it to me : francois@clickteam.com
Of course, the source code of the iOS runtime is NOT public, or in any kind of GNU licence. It remains the property of Clickteam, and you should not copy it or distribute it in any way.

Getting ready to program.

XCode.

The first thing you need to do, is to copy the file name "RuntimeIPhone.zip" on your Mac and unzip it. Then open the project in XCode. You are ready to begin development.

Creating a new extension.

To create a new extension :

- Right click on the "**Extensions**" folder in the project window.
- Choose "**New file**"
- Choose "**Objective-C class** and click on Next
- Enter for the name of your extension, the name "**CRun**" followed by the file name of your extension (the name is case sensitive). For example if the extension name is kchisc then the name of the class should be "CRunkchisc"
- Start programming your extension.

You will have to incorporate your extension in the main iOS runtime. So do so :

- Open the file "Extensions / CExtLoad.m"
- At the beginning of the file, between the F01 and F01END markers, insert a line to import the .h file of your extension. In our example, it would be: #import "Runkchisc.h"
- In the loadRunObject method, between the F02 and F02ENDmarkers, insert the following code :

```
if ([name caseInsensitiveCompare:@"name_Of_Extension"==0)
{
        object=[[CRunName_Of_Extension alloc] init];
}
```

This line will incorporate your extension in the runtime. The name in quotes must be the name of the extension as it is on disc (in our example, kchisc).

Your extension must reside in one .m and .h file. You are free to create more than one

classes in these sources.

Please note that the sources of MMF Runtime included in the SDK contain the sources of all the extension made at that time. If you need to see example extensions, just have a look at all the files in the extensions folder.

Running applications with your extension.

To be able to debug your extensions in XCode, you need a version of MMF2 with the iOS exporter. Create your test application, and select iOS application as a build type. Then build your application in the RuntimeIPhone folder on your Mac, replacing the current Application.cci Then compile and run your application to debug it.
To be able to use your extension, you need an MMF2 whose build number is 252 at least.

Publishing your extension

Once you are happy with your extension, you need to publish it. To do so :
- Compress the RuntimeIPhone folder on your Mac (by right clicking on the folder and choosing Compress RuntimeIPhone
- Copy the RuntimeIPhone.zipfile on your PC
- Start iOSDatMaker (found in the root of the iOS SDK folder
- Select the RuntimeIPhone.zipfile
- Enter the name of your extension (respecting case)
- Choose the destination directoy
- The program will then create a .ext file containing the sources of your extension. Just copy it in your MMF2 Data\Runtime\iPhone folder. After that, your extension should be available when building applications.

The extension class

As stated before, the name of this class should be "CRunNameOfExtension". It should be in the Extension folder.
The class is derived from an class, named CRunExtension.
The class contains functions similar to the C++ extensions, for the runtime. Only more structured. Lets see the functions in details.
In MMF2 iOS runtime, all object's name start with "C".

The minimal #import necessary in the .m file for your extension to compile are the following:
#import “Extension.h”
#import “Run.h”
#import “Bitmap.h”
#import “Services.h”
#import “CreateObjectInfo.h”
#import “Image.h”
#import “ActExtension.h”
#import “CndExtension.h”
#import “Value.h”

Default variables

Two variables are defined in the class parent from the object :
- ho : points to the CExtension object. Equivalent to the headerObject structure in MMF. Ho is also useful as callback functions are defined to exchange data with the main runtime.
- rh : points to the CRun object. Equivalent to the runHeader structure in MMF.

Constructor
Nothing special to do in the constructor of the extension. But you can have your own code in it.

-(int)getNumberOfConditions

This function should return the number of conditions contained in the object (equivalent to the CND_LAST define in the ext.h file).

-(BOOL)createRunObject(CFile):file withCOB:(CCreateObjectInfo)cob andVersion: (int)version

This function is called when the object is created. As iOS object are just created when needed, there is no EDITDATA structure. Instead, I send to the function a CFile object, pointing directly to the data of the object (the EDITDATA structure, on disc).
The CFile object allows you to read the data. It automatically converts PC-like ordering (big or little Indian I cant remember) into iOS ordering. It contains functions to read bytes, shorts, int, colors and strings. Read the documentation about this class at the end of the document.
"Version" contains the version value contained in the extHeader structure of the EDITDATA.

So all you have to do, is read the data from the CFile object, and initialise your object accordingly. Return YES if your object has been created successfully.

-(void)destroyRunObject:(BOOL)bFast

Called when the object is destroyed. This routine should free all the memory allocated during createRunObject. bFast is true if the object is destroyed at end of frame. It is false if the object is destroyed in the middle of the application.

-(int)handleRunObject

Same as the C++ function. Perform all the tasks needed for your object in this function. As the C function, this function returns value indicating what to do :
- REFLAG_ONESHOT : handleRunObject will not be called anymore
- REFLAG_DISPLAY : displayRunObject is called at next refresh.
- Return 0 and the handleRunObject method will be called at the next loop.

-(void)displayRunObject:(CRenderer*)renderer

Called to display the object. If your object needs to be displayed on the screen, you need to add it to one of the display lists.
Please refer to the Crenderer class documentation at the end of this file.

-(void)pauseRunObject

Called when the application goes into pause mode.

-(void)continueRunObject

Called when the application restarts.

-(CfontInfo)getRunObjectFont

Equivalent to the C++ version. This function returns a CFontInfo object, a small object equivalent to the LOGFONT structure in C. See at the end of the document the definition of the CFontInfo object.

-(void)setRunObjectFont:(CFontInfo)fi withRect:(CRect)rc

Called when the font of the object needs to be changed. The rc parameter is null when no resize is needed.

-(int)getRunObjectTextColor

Returns the current color of the text as an Integer.

-(void)setRunObjectTextColor:(int)rgb

Sets the current color of the text.

-(Cmask*)getRunObjectCollisionMask:(int)flags

If implemented, this function should return a CMask object that contains the collision mask of the object. Return null in any other case.

-(BOOL)condition:(int)num withCndExtension:(CCndExtension*)cnd

The main entry for the evaluation of the conditions.
- num : number of the condition (equivalent to the CND_ definitions in ext.h)
- cnd : a pointer to a CCndExtension object that contains useful callback functions to get the parameters.

This function should return YES or NO, depending on the condition.

-(void)action:(int)num withActExtension:(CActExtension*)act

The main entry for the actions.
- num : number of the action, as defined in ext.h
- act : pointer to a CActExtension object that contains callback functions to get the parameters.

-(CValue*)expression:(int)num

The main entry for expressions.
- num : number of the expression

To get the expression parameters, you have to call the getExpParam method defined in the "ho" variable, for each of the parameters. This function returns a CValue* which contains

the parameter. You then do a getInt(), getDouble() or getString() with the CValue object to grab the actual value.
This function returns a pointer to a Cvalue object containing the result. The content of the CValue can be a integer, a double or a String. There is no need to set the HOF_STRING flags if your return a string : the CValue object contains the type of the returned value.

You should not alloc the CValue yourself, but instead ask for a temporary value from the Crun class :
[rh getTempValue:default_value] where default_value is an integer.

Callback functions

The "ho" variable in the extension object gives you access to the CExtension object, which is a derivative of the main CObject class. I have programmed a few callback functions in the CExtension object. Here are these functions :

-(int)getX

Returns the current X co-ordinate of the object (hoX).

-(int)getY

Returns the current Y coordinate of the object (hoY)

-(int)getWidth

Returns the current width of the object (hoImgWidth).

-(int)getHeight

Returns the current height of the object (hoImgHeight).

-(void)setPosition:(int)x withY:(int)y

Changes the position of the object, taking the movement into account (much better than poking into hoX and hoY).

-(void)setX:(int)x

Changes the position of the object (hoX), and takes care of the movement and refresh. Same remark as setPosition.

-(void)setY:(int)y

Same as setX for Y co-ordinate. Same remark as setPosition.

-(void)setWidth:(int)width

Change the width of the object, taking care of the hoRect fields.

-(void)setHeight:(int)height

Same as setWidth, for height.

-(void)loadImageList:(short*)imageList withLength:(int)length

This method should be called in the createRunObject method of your object. If your object uses images stored in the image bank, you must call this method so that the proper images are loaded.
Just make an array with all the handles of the images, the size should be the exact number of images to load. Call this method (it may take some time to return). All the images will be loaded in the runtime.

-(CImage*)getImage:(short)handle

Call this function to retrieve an image from a handle. The image must have been previously loaded with loadImageList. The value returned could be nil if the image could not be loaded, but this is very unlikely to occur.
CImage is part of the "Banks" package. So you should import it with "#import 釘 anks.h".

-(void)reHandle

If you returned a REFLAG_ONESHOT value in your handleRunObject method, this will reinforce the method to be called at each loop.

-(void)generateEvent:(int)code withParam:(int)param

Generate an event with the specific code. The parameter can be recuperated with the [ho getEventParam] method. You should prefer the push_event method, specially if your event is generated in another thread than the main thread.

-(void)pushEvent:(int)code withParam:(int)param

This is the method of choice to generate an event.

-(void)pause

Pauses the application, when you have some lengthy work to perform (like opening a file selector).

-(void)resume

Resumes the application at the end of your work.

-(void)redraw

Forces a redraw of the object (the displayRunObject routine is called at next refresh).

-(void)destroy

Destroys the object at the end of the current loop.

-(int)getExtUserData

Returns the private field of the extHeader structure.

-(void)setExtUserData:(int)data

Changes the private field of the extHeader structure.

-(int)getEventCount

Returns the rh4EventCount value, used in controls to trigger the events.

-(CValue*)getExpParam

Returns the next expression parameter.

-(CObject*)getFirstObject

Returns the first object currently defined in the frame. Should be used in conjunction with getNextObject().

-(CObject*)getNextObject

Returns the next object in the list of objects of the frame. Returns nil if no more objects is available. This method will return the extension object it is called from.

CRun callback functions

You access the CRun object via the "rh" variable defined in the CRunExtension class. This object contains three methods used to define global data.

Some extensions need to communicate between objects. In C++ this was done simply by defining global variable in the code. I have defined three functions in CRun to allow you to create global classes.

-(void)addStorage:(CExtStorage*)data withID:(int)id

Adds a new object to the storage, with the "id" identifier. "id" is a simple integer number. The storage class must be derived from the class CExtStorage. This function has no effect if an object with the same identifier already exists.

-(CextStorage*)getStorage:(int)id

Returns the storage object with the given identifier. Returns nil if the object is not found.

-(void)delStorage:(int)id

Deletes the object with the given identifier.

CActExtension callback functions

The CActExtension object is transmitted to the extension "action" method when a action is called. This object contains callback function to gather the parameters of the action.
These functions want two parameters:
- The CRun object (available in the extension as "rh").
- The number of the parameter in the action, starting at 0

-(CObject*)getParamObject:(CRun*)rhPtr withNum:(int)num

Returns the CObject pointed to by the PARAM_OBJECT.

-(int)getParamTime:(CRun*)rhPtr withNum:(int)num

Returns the time value in milliseconds.

-(int)getParamBorder:(CRun*)rhPtr withNum:(int)num

Returns the border parameter.

-(int)getParamDirection:(CRun*)rhPtr withNum:(int)num

(obsolete, but might be used in old extensions). Returns a direction from 0 to 31.

-(int)getParamAnimation:(CRun*)rhPtr withNum:(int)num

Returns the number of the animation.

-(int)getParamPlayer:(CRun*)rhPtr withNum:(int)num

Returns the number of the player.

-(LPEVP)getParamEvery:(CRun*)rhPtr withNum:(int)num:PARAM_EVERY

Returns a pointer to the eventParam structure containing the Every parameter. Look for the definition in Events.h.

-(int)getParamSpeed:(CRun*)rhPtr withNum:(int)num

Returns a speed, from 0 to 100.

-(unisgned int)getParamPosition:(CRun*)rhPtr withNum:(int)numCPositionInfo

Returns the X and Y position contained in the parameter. Use the macros HIWORD(parameter) to get the X position, and LOWORD(parameter) to get the Y position.

-(int)getParamJoyDirection:(CRun*)rhPtr withNum:(int)num

Returns a joystick direction.

-(short*)getParamZone:(CRun*)rhPtr withNum:(int)num

Returns a pointer to the PARAM_ZONE parameter. The parameter is defined like this:
short X1;
short Y1;
short X2;
short Y2;

-(int)getParamExpression:(CRun*)rhPtr withNum:(int)num

Returns the value contained in the expression.

-(int)getParamColour:(CRun*)rhPtr withNum:(int)num

Returns a color, as an integer.

-(int)getParamFrame:(CRun*)rhPtr withNum:(int)num

Returns a number of frame.

-(int)getParamNewDirection:(CRun*)rhPtr withNum:(int)num

Returns a direction, from 0 to 31.

-(int)getParamClick:(CRun*)rhPtr withNum:(int)num

Returns the click parameter (left/middle/right button).

-(NSString*)getParamExpString:(CRun*)rhPtr withNum:(int)num

Returns the result of a string expression. You should copy he string as it is likely to be destroyed by MMF.

-(double)getParamExpDouble:(CRun*)rhPtr withNum:(int)num

Returns as a double the result of the evaluation of the parameter.

CCndExtension callback functions

The CCndExtension object is transmitted to the extension when calling the condition method. It contains callbacks to gather the parameters of the condition. Most of the method are identical to the ones in CActExtension, with the following differences :

Missing methods:
getParamPosition

Different returns
The getParamObject method returns a pointer to the EventParam structure (as in the C++).
-(LPEVP)getParamObject:(CRun*)rhPtr withNum;(int)num

-(BOOL)compareValues:(CRun*)rhPtr withNum:(int)num andValue:(CValue*)value

In the C++ version, when you had a PARAM_COMPARAISON parameter in the condition, the condition routine returned a long value, and MMF was automatically doing the comparison with the parameter. You have to call this method in the iOS version. For example, a condition with a PARAM_COMPARAISON as first parameter, the end of the condition method should be:

return [cnd compareValues:rh withNum:0 andValue:returnValue];

Where returnValue is a CValue object containing the value to compare with.

-(BOOL)compareTime:(CRun*)rhPtr withNum:(int)num andTime:(int)t

Same as the previous function, for PARAM_CMPTIME parameters, where "t" is the time to compare to.

Useful objects

I will document now some of the objects of the runtime you will be using to program your extension.

The CFile object

This object is sent to you in the createRunObject method. It automatically points to the start of the extension data (EDITDATA structure, in memory). It automatically performs the indian translation between PC values and Flash values.

-(void)readACharBuffer:(char*)pBuffer withLength:(int)length

Read a array of char.

-(void)readAUnicharBuffer:(unichar*)pBuffer withLength:(int)length

Read a array of unichar.

-(void)skipBytes:(int)n

Skips bytes in the file.

-(int)getFilePointer

Returns the current file pointer position.

-(void)seek:(int)pos

Change the file pointer position.

-(unsigned char)readAByte

Reads one byte.

-(short)readAShort

Reads one short (two bytes).

-(int)readAInt

Reads an integer (4 bytes).

-(char)readAChar

Reads an char (2 bytes).

-(int)readAColor

Reads a color making it compatible with iOS color values (inversion of Red and Blue values).

-(NSString*)readAString

Reads a string that finishes with 0. The string is allocated for you, you should release it later in the code.

-(NSString*)readStringWithSize:(int)size

Reads a string of the given size. The string is allocated for you, you should release it later in the code.

-(CFontInfo*)readLogFont

Reads a LOGFONT structure into a CFontInfo object. The object is allocated for you, you should release it later in the code.

-(CFontInfo*)readLogFont16()

Reads a LOGFONT structure into a CFontInfo object in the old 16 bits format. The object is allocated for you, you should release it later in the code.

The CValue object

The CValue object is used in expression evaluation. It can contain an Integer, a double or a NSString.

-(id)initWithInt:(int)i

Creates the object as type integer with the given value.

-(id)initWithDouble:(double)d

Creates the object as type double with the given value.
-(id)initWithString:(NSString*)string

Creates the object as type integer with the given value.

-(short)getType

Returns the type of the object. The return value can be :

TYPE_INT : and integer
TYPE_DOUBLE : a double
TYPE_STRING : a string

-(int)getInt

Returns an integer. (if the object is of TYPE_DOUBLE, converts the value to int).

-(double)getDouble

Returns a double.

-(NSString*)getString

Returns the string. Warning, if the object is of type int or double, this will NOT convert the value to string.

-(void)forceInt:(int)value

Forces the value of type int into the object (if the object was previously double or int, then changes the type).

-(void)forceDouble:(double)number

Forces the value of type double.

-(void)forceString:(NSString*)string

Forces a string.

-(void)forceValue:(CValue*)value

Forces the content and type of the CValue.

-(void)setValue:(CValue*)value

Change the content of the object, respecting its type.

The CFontInfo object

The CFontInfo object is a replacement of the LOGFONT structure in C++. It contains the name of the font, its height, weight and attributes.

Fields contained in the object :

```
int lfHeight;
int lfWeight;
unsigned char lfItalic;
unsigned char lfUnderline;
unsigned char lfStrikeOut;
NSString* lfFaceName;
```

The CRect structure

This structure is intended as a replacement of the C++ RECT structure.

Fields :

```
int left;
int top;
int right;
int bottom;
```

C-type functions are defined for CRect :

CRect CrectLoad(CFile* file)

Loads a Crect structure from a Cfile.

CRect CrectInflate(CRect rc, int dx, int dy)

Grows the size of the given CRect.

CRect CRectNil()

Returns an empty CRect.

BOOL CRectAreEqual(CRect a, CRect b)

Returns YES if the two given CRect are identical.

BOOL CRectPointInRect(CRect rc, int x, int y)

Returns YES if the point lies within the CRect surface.

BOOL CRectIntersects(CRect a, CRect b)

Returns YES if the two CRect intersetcs.

The CPoint structure

The CPoint structure is a replacement of the C++ POINT structure. Is only contains two

fields:

```
int x;
int y;
```

The CRenderer class

COMMON FUNCTIONS//

-(void)clear:(float)red green:(float)green blue:(float)blue;

Clears the current render target (either the screen or the currently bound CRenderToTexture)

-(void)setClipWithX:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h;

Sets the Scissor rectangle to begin at 'x', 'y' with the given 'width' and 'height'.
No drawing can be performed outside the currently set scissor rectangle until it is reset again with a call to 'resetClip'.

-(void)resetClip;

Resets the current scissor rectangle and drawing is allowed over the entire screen or CRenderToTexture.

DRAWING

-(void)renderSimpleImage:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h;

Renders the previously set image (either CRenderToTexture or CImage) to the given x,y coordinate with the width and height.
Note: This method only reuses whatever previously set drawing settings (ink-effect, blending color and the like).
This is useful if you want to batch-draw a whole set of images using the same settings and only want the position/size to vary.

This routine is for example used when drawing the tiled images in a mosaic quick backdrop.

-(void)renderImage: (ITexture*)image withX:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h andInkEffect:(int)inkEffect andInkEffectParam:(int)inkEffectParam;

The most common image drawing rountine that takes an image, x,y,width,height and ink-effect parameters and draws the image there immedeatly.
This routine does not take hot-spots into account so the hotspot must be subtracted from the position before passing the position on to this routine.

Ink effect values:

The inkEffect variable contains both the ink-effect to use and some flags.
You can set the ink-effect to these values:
BOP_BOPY

BOP_BLEND
BOP_INVERT
BOP_ADD
BOP_SUB

If the BOP_RGBAFILTER flag is not set in the inkEffect variable then the effectParam value will be treated as a blending value between 0 and 128.
If the BOP_RGBAFILTER flag is set, then the effectParam value should contain an ARGB color. (Alpha first, then RGB value f

Example 1: Draw an image using 50% semitransparency
[renderer renderImage:image withX:0 andY:0 andWidth:32 andHeight:32 andInkEffect:BOP_BLEND andInkEffectParam:64];

Example 2: Draw an image with a red blending color and 50% semitransparency
[renderer renderImage:image withX:0 andY:0 andWidth:32 andHeight:32 andInkEffect:BOP_BLEND|BOP_RGBAFILTER andInkEffectParam:getARGB(128,255,0,0)];

Note: getARGB() requires that CServices.h is imported.

-(void)renderScaledRotatedImage:(ITexture*)image withAngle:(int)angle andScaleX:(float)sX andScaleY:(float)sY andHotSpotX:(int)hX andHotSpotY:(int)hY andX:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h andInkEffect:(int)inkEffect andInkEffectParam:(int)inkEffectParam;

Works the same way as 'renderImage' with a few key differences: It takes an angle and a hotSpot position around which the image will be rotated.

-(void)renderPattern: (ITexture*)image withX:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h andInkEffect:(int)inkEffect andInkEffectParam:(int)inkEffectParam;

Draws a pattern of the given 'image' starting at (x,y) and will be repeated continously inside the width and height.
The (x,y) denotes the top-left corner of the rectangle inside which it will draw.

Note: See 'renderImage' for an explanation of how to use the ink-effects.

-(void)renderPatternEllipse: (ITexture*)image withX:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h andInkEffect:(int)inkEffect andInkEffectParam:(int)inkEffectParam;

Same as 'renderPattern' but is masked out in the shape of an ellipse that fits inside the width and height.
The (x,y) denotes the top-left corner of the rectangle inside which it will draw.
See 'renderImage' for an explanation of how to use the ink-effects.

-(void)renderPoint: (ITexture*)image withX:(int)x andY:(int)y andInkEffect:(int)inkEffect andInkEffectParam:(int)inkEffectParam;

Identical to 'renderImage' except it only draws a single 1x1 pixel. Gives a speedup when many of theese are drawn.

-(void)renderLineWithXA: (int)xA andYA:(int)yA andXB:(int)xB andYB:(int)yB andColor:(int)color andThickness:(int)thickness;

Draws a line from (xA,yA) to (xB,yB) with the given color and thickness.
Note: Colors are in RGBA format, not ARGB.

-(void)renderGradient:(unsigned char*)colors withX:(int)x andY:(int)y andWidth: (int)w andHeight:(int)h andInkEffect:(int)inkEffect andInkEffectParam: (int)inkEffectParam;

Draws a rectangular gradient from (x,y) to (x+w, y+h) using the given ink-effect.
The 'colors' array should contain 4 RGBA colors (16 unsigned chars) each color being one of the corner colors.

Note: The order of the colors are as follows:
top-left, top-right, bottom-left, bottom-right (follows a Z pattern).

Note: See 'renderImage' for an explanation of how to use the ink-effects.

-(void)renderGradientEllipse:(unsigned char*)colors withX:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h andInkEffect:(int)inkEffect andInkEffectParam: (int)inkEffectParam;

Draws an ellipse shaped gradient contained within the rectangle that goes from (x,y) to (x+w, y+h) using the given ink-effect.
The 'colors' array should contain 4 RGBA colors (16 unsigned chars) each color being one of the corner colors in the bounding rectangle.

Note: The order of the colors are as follows:
top-left, top-right, bottom-left, bottom-right (follows a Z pattern).

Note: See 'renderImage' for an explanation of how to use the ink-effects.

-(void)renderSolidColor:(int)color withX:(int)x andY:(int)y andWidth:(int)w andHeight:(int)h andInkEffect:(int)inkEffect andInkEffectParam:(int)inkEffectParam;

Draws a simple RGBA color within the rectangle from (x,y) to (x+w, y+h).

MANUAL FUNCTIONS

-(void)flush;

This function flushes the OpenGL rendering queue and waits for it to finish doing so.
It is not adviced to use this in a real-time rendering loop but can be useful if you want to be sure OpenGL is done rendering something before you for example start a timer.

-(void)useBlending:(bool)useBlending;

Turns ON or OFF OpenGL blending mode. Turning this off effectively disables any kind of semi-transparency/alpha blending.
This can be used when you want to render to a texture in an 'overwrite' fashion.

-(void)uploadTexture:(ITexture*)texture;

Uploads a given texture (CRenderToTexture or CImage) to the graphics card if nessecary.
Note: All the drawing commands automatically checks if the texture is resident on the graphics card before drawing so you should normally not need to call this.

-(void)removeTexture:(ITexture*)texture;

Deletes the texture from the graphics card to free up memory.

-(void)updateViewport;

Sets the current viewPort (and therefore defines the extent of the local coordinate system). Normally this should just be set to the size of either the screen or the size of the CRenderToTexture.

-(void)setTexture:(ITexture*)texture;
-(void)setVertices:(GLfloat*)vertices;
-(void)setTexCoords:(GLfloat*)texCoords;
-(void)setColors:(unsigned char*)colors;
-(void)setInkEffect:(int)effect andParam:(int)effectParam;

This sets the current drawing settings of the drawing system.
Subsequent calls to 'renderSimpleImage' will use these settings except the vertices.

setVertices and setTexCoords expects an array containing 4 coordinates (8 floats X,Y) in a Z-pattern:
top-left, top-right, bottom-left, bottom-right.

The texture coordinates should be in the [0..1] range.

setColors expects an array containing 4 RGBA colors (16 unsigned chars) - also in a Z-pattern.

Note: See 'renderImage' for an explanation of how to use the ink-effects.

-(void)cleanMemory;

Clears all uploaded textures from the graphics card.

-(void)setProjectionMatrix:(int)x andY:(int)y andWidth:(int)width andHeight: (int)height;

Sets the orthogonal projection matrix of OpenGL. The bottom-right corner of the current openGL viewport will have the coordinate (width,height).
Normally in the MMF2 runtime the coordinate of this point matches the size of the viewport so there is always a 1:1 ratio between the coordinates and pixels.

TRANSITIONS

-(void)setOriginX:(int)x andY:(int)y;

Sets the origin of the transition blitting routines in the destination buffer (screen or CRenderToTexture)
All transition blitting routines will be offset by this amount.

-(void)renderBlitFull: (CRenderToTexture*)source;

Blits the entire CRenderToTexture into the currently bound destination (screen or another CRenderToTexture)
Note: You cannot blit from a CRenderToTexture into itself.

-(void)renderFade: (CRenderToTexture*)source withCoef:(int)alpha;

Same as 'renderBlitFull' but takes an alpha parameter defining the alpha transparency of the blit.

-(void)renderBlit: (CRenderToTexture*)source withXDst:(int)xDst andYDst:(int)yDst andXSrc:(int)xSrc andYSrc:(int)ySrc andWidth:(int)width andHeight:(int)height;

Blits the 'source' image from the rectangle (xSrc,ySrc)(xSrc+width, ySrc+height) into (xDst,yDst)(xDst+width, yDst+height) on the screen or in the bound CRenderToTexture.
Note: You cannot blit from a CRenderToTexture into itself.

-(void)renderStretch: (CRenderToTexture*)source withXDst:(int)xDst andYDst: (int)yDst andWDst:(int)wDst andHDst:(int)hDst andXSrc:(int)xSrc andYSrc:(int)ySrc andWSrc:(int)wSrc andHSrc:(int)hSrc andInkEffect:(int)inkEffect andInkEffectParam:(int)inkEffectParam;

Blits the source image from the rectangle (xSrc,ySrc)(xSrc+wSrc, ySrc+hSrc) into the rectangle (xDst,yDst)(xDst+wDst, yDst+hDst) on the screen or in the bound CRenderToTexture using the given ink-effect.

Note: You cannot blit from a CRenderToTexture into itself.
Note: See 'renderImage' for an explanation of how to use the ink-effects.

-(void)renderStretch: (CRenderToTexture*)source withXDst:(int)xDst andYDst: (int)yDst andWDst:(int)wDst andHDst:(int)hDst andXSrc:(int)xSrc andYSrc:(int)ySrc andWSrc:(int)wSrc andHSrc:(int)hSrc;

Blits the source image from the rectangle (xSrc,ySrc)(xSrc+wSrc, ySrc+hSrc) into the rectangle (xDst,yDst)(xDst+wDst, yDst+hDst) on the screen or in the bound CRenderToTexture.

Note: You cannot blit from a CRenderToTexture into itself.

The CrenderToTexture class

This class is what can be compared to the cSurfaces in the MMF2 standard SDK except

that these images reside completely on the graphics card.

-(id)initWithWidth:(int)w andHeight:(int)h andRunApp:(CRunApp*)runApp;

Initializes the object with the given size. The texture is automatically resized up to the nearest power of two size (but this is transparent to the user of the class)

Usage:
CRenderToTexture* mySurface = [[CRenderToTexture alloc] initWithWidth:128 andHeight:128 andRunApp:rhApp];

-(void)bindFrameBuffer;

Binds this CRenderToTexture to be the destination for all future drawing calls. Remember to unbind again after use or nothing will ever get drawn to the screen.

-(void)unbindFrameBuffer;

Unbinds the CRenderToTexture and restores the previously bound buffer (can be another CRenderToTexture or the screen)

-(void)fillWithColor:(int)color;

Clears the color channels (RGB) with the given color

-(void)clearWithAlpha:(float)alpha;

Clears the surface and sets the alpha channel to the given value

-(void) clearAlphaChannel:(float)alpha;

Keeps the RGB colors but sets the alpha channel to the given value

-(int)uploadTexture;

Does nothing for CRenderToTexture objects as they are always residing on the graphics card.

-(int)deleteTexture;

Does nothing. The texture is deleted when the CRenderToTexture object itself is released.

-(int)getTextureID;

Returns the OpenGL texture ID of the CRenderToTexture

-(int)getWidth;

Returns the width of the surface (the width not rounded up to a power-of-two)

-(int)getHeight;

Returns the heigth of the surface (the height not rounded up to a power-of-two)

-(int)getTextureWidth;

Returns the width of the texture (power-of-two)

-(int)getTextureHeight;

Returns the height of the texture (power-of-two)

-(float*)getTextureCoordinates;

Returns an array pointer that contains the texture coordinates for the image.

The CtextSurface class

A buffer class to prevent text from being drawn constantly by buffering it inside a CImage class. Use this class whenever you want to draw text to the screen.

There are two modes of drawing text with this class: 'Single string', and 'manual'.

Single string:

If you only want to draw a single string somewhere in your extension you can use this method as it is far the simplest.

In your extensions' drawing routine you simply call the 'setText' routine on the CTextSurface with the text, flags, color and font. CTextSurface will itself check if it is necessary to redraw the text in the buffer and upload it to the graphics card automatically. After the string is set you can simply call 'draw' with the given position and ink-effect to draw it wherever you like.

Manual:
If your extension draws multiple strings at one time (for example the Date & Time object in analog clock mode draws 12 strings in a circular fashion) the manual mode will be the right for you. It involves a little more work for your part:
- You first have to manually clear the text surface using 'manualClear'
- You then call 'manualDrawText' as many times as you like to draw text wherever you want.
- You must finally call 'manualUploadTexture' before you call the 'draw' method.

Note: This mode does not automatically detect whenever it is necessary to update and re-upload the text to the graphics card so you must try to keep this to a minimum. For example the Date & Time object only updates the text whenever it is necessary (typically only once at the creation time of the object).

-(id)initWidthWidth:(int)w andHeight:(int)h;

Initializes the CTextSurface with the given size.

Usage:

CTextSurface* myText = [[CTextSurface alloc] initWithWidth:200 andHeight:150];

-(void)setText:(NSString*)s withFlags:(short)flags andColor:(int)color andFont: (CFont*)font;

Sets the current text, flags, color and font of the CtextSurface. It only reuploads the text when absolutely nessecary so you can call this rountine as many time as you want with the same options without causing an expensive text-upload. Very useful for porting simple text displaying extensions.

-(void)draw:(CRenderer*)renderer withX:(int)x andY:(int)y andEffect:(int)inkEffect andEffectParam:(int)inkEffectParam;

Draws the text-buffer at (x,y) with the given ink-effect. See 'renderImage' in the documentation for the CRenderer class for how to use the ink-effects.

-(BOOL)setSizeWithWidth:(int)w andHeight:(int)h;

Resizes the text surface if needed. This routine clears any text drawn to the surface.

-(void)manualDrawText:(NSString*)s withFlags:(short)flags andRect: (CRect)rectangle andColor:(int)color andFont:(CFont*)font;

Manually draws the text in the given rectangle with the given color, font and flags.

-(void)manualUploadTexture;

When all drawing is done, upload the texture to the graphics card ready to be drawn using the 'draw' routine.

-(void)manualClear:(int)color;

Clears the drawn buffer ready for new text.

Objects using the CTextSurface (for reference):

- String object
- Score object
- Counter object
- Active and Background System Box
- Time and Date object (for the visual analog and digital clock)
- Hi-Score object