xfilesystem: Guide

**Virtuos Games**

**Monday, April 11, 2011**

Jurgen

**A description of xfilesystem**

# Interface

## xfilesystem

### init

For initializing xfilesystem you will need to implement an instance of xfilesystem::xthreading and supply an instance of xcore::x\_iallocator. The user also needs to supply a value for the maximum number of open streams.

### Exit

Upon exit all allocated objects will be deallocated and all registered xdevicealias objects will be invalidated.

## xdevicealias

To uniformly (cross-platform) access drives and locations this object will ‘map’ a device to a final system device plus path. This allows a game application to define an alias like ‘cache’ which maps to a device and/or location where to cache files that are needed in the game. On the WII this can be NAND, and on the Xbox 360 and PS3 this can be the harddisk. You could even use a memory file device to act as a cache.

Note: *An application is allowed to register a maximum of 64 aliases; however the user can replace existing ones.*

Most platforms register an average of 5 aliases upon initialization:

1. curdir
2. appdir
3. host
4. dvd
5. hdd

Note: *Win32 will register all existing FIXED, REMOVABLE, REMOTE and CDROM drive letters between ‘a’ and ‘z’.*

The game application can register an alias for the location of the game data, here is an example:

xfilesystem::xdevicealias gamedata ("data", "dvd");

xfilesystem::xdevicealias::sRegister(gamedata);

And from that moment on the application can open and use files. Here an example:

xfilepath fp("data:\\texture.tga");

xfilestream fs(fp, FileMode\_Open, FileAccess\_Read, FileOp\_Sync);

if (fs.isOpen())

{

xbyte\* buffer = new xbyte[fs.getLength()];

fs.read(buffer, 0, fs.getLength());

fs.close();

}

## xfiledevice

The user is allowed to implement and supply a custom xfiledevice derived instance. To do that the user needs to register an xdevicealias, here an example:

xfiledevice\* customHttpBasedFileDevice;

xfilesystem::xdevicealias::sRegister(xfilesystem::xdevicealias("fileserver", customHttpBasedFileDevice, ""));

## xstream

This is the public (strategy pattern) class that uses an instance of xistream. Any derived class of xstream is a ‘constructor’ class and is responsible for instantiating an instance of xistream. A xfilestream object can be cast to a xstream object without any danger, casting from xstream to xfilestream is not possible.

## xistream

This is the base class that defines the interface of any stream implementation; it is a near 1 on 1 copy of xstream. If the user wants to implement a new stream type he needs to implement a class derived from xistream. This object is a reference counted object.

## xfilestream

This is a constructor class that will obtain an instance of the private xifilestream class.

## xdirpath

This is a utility class for working with a directory path **without** file name and extension. It does correct the path string by always making sure that:

1. The path ends with a slash character
2. Slash characters are according to the platform, ‘\’ or ‘/’

## xfilepath

This is a utility class for working with directory path **with** file name and extension. It does correct the path string by always making sure that:

1. The path never ends with a slash character
2. Slash characters are according to the platform, ‘\’ or ‘/’

## xdirinfo

xdirinfo is an object that provides the user with functionality to manipulate a directory.

* Exists
* Create
* Remove
* Copy/Move
* Enumerate files and directories
* Time (Creation, Last Access, Last Write)

## xfileinfo

xfileinfo is an object that provides the user with functionality to manipulate a file.

* Exists
* Create, Create(xfilestream& outFileStream)
* Remove
* Read/Write
* Copy/Move
* Time (Creation, Last Access, Last Write)

# Threading

## XTHREADING

The main IO logic of xfilesystem should be called from a unique thread, the IO thread. The user is responsible of setting up this thread and calling xfilesystem::doIO(). To make this work the user also needs to implement an instance derived from xfilesystem::xthreading. It is however possible to make xfilesystem fully single threaded and thus all async IO will block the main thread, this is how it is done by the unittest.

## xasyncresult

# Memory

All necessary objects are allocated when calling xfilesystem::init, all other public objects upon usage do not allocate memory from the heap.

# A NOTE ON THE FILE CACHE

To provide the user with a file cache that uses NAND on WII, memory or the hard disc on PS3 and Xbox 360, it is advisable to create a new xcode package under the name xfilecache that uses the xfilesystem package to register a new ‘cache’ alias with the necessary implementation of a file device.