BaseLib2 Tutorial Series

Streams

André Neto

June 14, 2011

Outline

Streams?

- Read and write information
  - Streams allow to abstract the consuming media
    - Can be a string, a file, a socket, a console...
- You should use the Level2 classes
  - FString, File, TCPSocket, UDPSocket, Console, SXMemory,...
- ...and not the BasicXYZ Level0 classes
  - BString, BasicUDPSocket, BasicTCPSocket, ...

Streamable functionality

- Defines the streaming functions (most inherit from StreamInterface)
    - Read and Write (complete and buffered)
        - With timeout
    - Printf
    - Tokenizer
    - Line reading
- The actual streaming implementation (file, socket, string) will then provide the extra required functionality
    - e.g. Open and Close for a file or a socket
- Streamable can also contain a collection of switchable streams
    - Not always implemented

Streamable.h

Most important functions are

```
bool Read( void* buffer, uint32 & size, TimeoutType msecTimeout);
bool Write( const void* buffer, uint32 & size, TimeoutType msecTimeout);
virtual int64 Size();
virtual bool Seek(int64 pos);
virtual int64 Position(void);
virtual bool GetToken( char * buffer, const char * terminator, uint32
maxSize,...);
virtual bool GetLine( Streamable & output, bool skipTerminators =True);
bool Printf(const char *format,...);
virtual bool AddStream(const char *name);
virtual bool RemoveStream(const char *name);
virtual bool Switch(const char *name);
```

Introduction

- Easy use and management of strings
  - Assignment
  - Comparison
  - Allocation and deallocation
  - Size
  - Printfing
- You should no longer need to use const char * very often
- The name of the header file is FString.h (in Level2)
- Being streamable can be *interchanged* with other streams

FString.h

- A streamable string
- Some operators redefined (=, +=, ==, !=)

Example: BaseLib2/Documentation/Tutorials/examples/FStringExample.cpp

```
//Create an initialised FString
FString str1 = "/a/path/to/somewhere/";
//Create an empty FString
FString str2;
//Copy the contents of str1 to str2
str2 = str1;
//Compare the values (can also use the !=)
if(str2 == str1){
...
//It is also possible to printf in a string
FString str3;
str3.Printf("The value is %f", 0.12345);
//To tokenize a string use the GetToken
FString token;
while(str1.GetToken(token, "/")){
...
//It is also possible to concatenate to strings
str1 += "yet/another/string";
...
```

InternetAddress.h

- Used by all Sockets implementations, enables to store and retrieve information about the hostname and port

Most important functions are

```
int16 Port();
const char *HostName(BString &hostName);
static const char *LocalAddress();
static const char *LocalIpNumber();
```

Example: BaseLib2/ Documentation/Tutorials/examples/LocalInternetAddressExample.cpp

```
CStaticAssertErrorCondition(Information, "This computer local address is:%s",
InternetAddress::LocalAddress());
CStaticAssertErrorCondition(Information, "This computer local ip is:%s",
InternetAddress::LocalIpNumber());
```

Sockets

- Available socket types for TCP, UDP and ATM
  - All inherit from Streamable (again use the full featured sockets from Level2, not the Basic from Level0)
- Can be used in server and client mode
- Possibility to set in blocking mode
- All inherit basic functionality from BasicSocket (see BasicSocket.h)

Most important functions are

```
bool SetBlocking(bool flag);
InternetAddress &Source();
InternetAddress &Destination();
```

TCP socket server mode

- Open the socket
  - `Open();`
- Set in server mode
  - `bool Listen(int port,int maxConnections=1);`
- If you wish, set it to be blocking
  - `bool SetBlocking(bool flag);`
- Wait for connections
  - `BasicTCPSocket *WaitConnection(TimeoutType msecTimeout = TTInfiniteWait,BasicTCPSocket *client=NULL);`
  - Notice that a new socket, with the client connection, is returned
  - You can now read and write to this socket just as you would do with any other stream
- Usually multi-threading is used to handle new connections

TCP socket server example (BaseLib2/Documentation/Tutorials/examples/SimpleTCPServer.cpp)

Example code

```
//Create a server running in port 12468
int32 port = 12468;
TCPSocket server;
//Open the socket
if(!server.Open()){
...
if(!server.Listen(port)){
...
TCPSocket *client = server.WaitConnection();
//Set the client in blocking mode for the read
client->SetBlocking(True);
//Print information from the client
FString hostname;
client->Source().HostName(hostname);
CStaticAssertErrorCondition(Information, "Accepted a connection from %s", hostname.Buffer());
//Read a line from the client socket
FString line; client->GetLine(line);
CStaticAssertErrorCondition(Information, "Read line from socket:  %s", line.Buffer());
```

TCP socket client mode

- Open the socket
    - `Open();`
- Connect to the server
    - `bool Connect(const char *address,int port,TimeoutType msecTimeout = TTInfiniteWait, int retry=12)`
    - You can now read and write to this socket just as you would do with any other stream
- Remember housekeeping... always close the sockets when you no longer need them

TCP socket client example (BaseLib2/Documentation/Tutorials/examples/SimpleTCPClient.cpp)

Example code

```
//Connect to a server running in localhost and in port 12468
int32 port = 12468;
FString host = "localhost";
TCPSocket client;
//Open the socket
if(!client.Open()){
...
//Connect to the server
if(!client.Connect(host.Buffer(), port)){
...
//Write a line
FString line = "Hello!";
uint32 size = line.Size();
if(!client.Write(line.Buffer(), size)){
...
//Notice that the actual number of bytes wrote is updated on the size variable
CStaticAssertErrorCondition(Information, "Successfully wrote %d out of %lld bytes to %s:%d",
size, line.Size(), host.Buffer(), port);
//Housekeeping
client.Close();
```

UDP sockets

- Usage very similar to the TCP socket, but obviously stateless
    - No WaitConnection in the server mode
    - Servers read and handle data directly from the socket (usually multi-threaded)

Snippet code

```
while(IsAlive()){
    read = 1024;
    if(serverSocket->Read(buffer, read)){
        ...//New data is available
```

File.h

- All streaming function available
- Open is based on read/write modes

Most important functions are

```
bool Open(const char *fname,...);
bool OpenRead(const char *fname,...);
bool OpenWrite(const char *fname,...);
inline bool OpenNew(const char *fname,...);
void SetOpeningModes(uint32 modeSet);
bool FileLock(BasicFile &f,int64 start,int64 size,TimeoutType
msecTimeout);
bool FileUnLock(BasicFile &f,int64 start,int64 size,TimeoutType
msecTimeout);
bool FileEraseFile(const char *fname,...);
```

File handling example (BaseLib2/Documentation/Tutorials/examples/SimpleFileHandling.cpp)

Example code

```
//Create an output file
FString filename = "output.txt";
File output;
if(!output.OpenWrite(filename.Buffer())){
...
//Write something to it
FString line = "Write something\ninto this\nfile\n";
uint32 size = line.Size();
if(!output.Write(line.Buffer(), size)){
...
//Housekeeping
output.Close();
//Open the file for reading
File input;
if(!input.OpenRead(filename.Buffer())){
...
//Reset the line string
line.SetSize(0);
//Read the file
while(input.GetLine(line)){
    CStaticAssertErrorCondition(Information, "Read:  %s", line.Buffer());
    line.SetSize(0);
}
input.Close();
```

Key ideas

- Universal console mechanism
  - Should be optimised for each OS console
  - Easy to set colours and if supported dimensions, titles, …
- Read and write just like any other stream
- Particularly useful to get user input
  - Special automatically built-in menus are available in other classes

Console example (BaseLib2/Documentation/Tutorials/examples/SimpleConsole.cpp)

Example code

```
//Create a console in single character read input
Console con(PerformCharacterInput);
con.Clear();
//Change the color of the text
con.Printf("Select foreground color\n");
PrintColourMenu(con);
char c1;
//Read a single char
con.GetC(c1);
switch(c1){
    case '1':
        con.SetColour(Red, Black);
        break;
    case '2':
        con.SetColour(Green, Black);
        break;
...
}
//Read a line
FString line;
con.Printf("\nWrite a line\n");
con.GetLine(line);
con.Printf("You wrote:  %s\n", line.Buffer());
```

Key ideas

- Add streaming capabilities to an existing buffer
- Very useful to use when the argument of a function is a streamable and we only have an anonymous buffer
- Most of the times the FString is a better option

Snippet code

```
void ReadFromStream(Streamable &stream){
...
}
...
float buffer[10];
SXMemory sxm((char *)&buffer[0], sizeof(float) * 10);
//the buffer can now be handled as a stream
ReadFromStream(sxm);
```

Training ideas

1. Write a network server with the following specifications
    1. Each client connection is handled in a different thread
    2. For each connection a new file, with an unique name (for you to decide), is open
        1. The first line of the file must contain the IP address and hostname of the client
    3. The connection is kept alive until the client sends a close keyword (like quit)
        1. or until the client closes the connection...
    4. Each line sent by the client is stored in this file
2. Write the client for the above server
    1. It should be capable of reading a file and sending its contents to the server
    2. and/or using a console, send each line input by the user
3. Write a simple program capable of copying the contents of two files