Subject: Re: About storing references and pointers to callbacks.
Posted by Oblivion on Mon, 26 Jun 2017 10:14:52 GMT
View Forum Message <> Reply to Message

Hello Mirek,

Sorry for the confusing code. That was not very informative, I admit.
They are not parts of a single file, they are separate, queued downloads.

Sftp class has both low level commands (corresponding to fopen, fstat, fread, fwrite) and their high level counterparts.
Below code simply demonstrates the low level code.

A real downloader with low level commands is like this: (usually we won't use these low level commands)

```
#include <Core/Core.h>
#include <SSH/SSH.h>

using namespace Upp;



CONSOLE_APP_MAIN
{
 const char *remote_path = "readme.txt";

 Ssh ssh;
 Cout() << "Connecting to Rebex public sftp test server...\n";

 // Let us connect in a blocking way.
 bool b = ssh.Connect("test.rebex.net", 22, "demo", "password");
 if(b) {
  // Note that below code respresents low level, async sftp file transfer.

       FileOut local_file("readme_downloaded.txt");

  // SFtp file attributes are stored in this structure, and used by low-level methods.
     SFtpAttrs file_attrs;

  // Now let us request a SFtp channel.
  SFtp sftp(ssh);

  // Chunk size is used only in gets and puts. Default chunk size is 32K
  //sftp.ChunkSize(16 * 1024);
```

```
  // Now let us queue the Sftp commands.
  sftp.StartInit();

   // Asynhronous calls can be queued. And they have both high level and low level
   // interface.
   // For example: StartGet has two variants:
   // 1) A low levet StartGet(): Requires a file handle (fopen) and file info (fstat) to operate on.
   //    So It requires StartOpen() and StartGetStat() methods to be called first.
   //    For this purpose, we can queue them and later run them:
                        // (Note that, any error while processing the commands will halt the queue, and
clean-up the mess. So it's safe.)

       sftp.StartOpen(remote_path, SFtp::READ, 0755);
      sftp.StartGetStat(file_attrs);
            sftp.StartGet(local_file);
            sftp.StartClose();

   // 2) A high level StartGet(): This method combines the above StartOpen, StartGetStat,
   // StartGet, and StartClose respectively, and handles errors internally:
   //
   // sftp.StartGet(local_file, remote_path, SFtp::READ, 0755);

  sftp.StartStop();

  // Now that we have queued up the download, using low level api, we can run the queue
asynchronously.
  // Note that we are running a single Sftp instance here.
  // We can do this with multiple instances (using an Array<SFtp>) and multiple commands too.

  while(sftp.Do()) {
   if(!sftp.InProgress()) {
    if(sftp.IsSuccess())
      Cout() << "File successfully downloaded.\n";
    else
     Cout() << "File download failed. Reason: " << sftp.GetErrorDesc() << "\n";
    break;
   }
  }
 }
 else
  Cout() << ssh.GetErrorDesc() << "\n";
 ssh.Disconnect();
}
```

The problem can arise when the loop is called elsewhere (where it may outlive the FileOut streams.). I guess it is simply enough to warn the users about this in the docs.