Subject: Re: FTP class and reference example for U++
Posted by Oblivion on Thu, 13 Apr 2017 06:17:33 GMT
View Forum Message <> Reply to Message

Hello Klugier,

Thank you very much for the feedback.

Quote:You could replace NAMESPACE_UPP with namespace Upp { and END_NAMESPACE_UPP with }. The NAMESPACE_UPP approach was deprecated since last release.

It seems that I missed the announcement. Will change this asap.

Quote:The next problem I see in the code is the ordering problem - you should put public class interface at the top then private things (Ftp and DirEntry classes). This is the general rule mentioned in following documentation topic.

I tend to follow the coding style of U++. It improves the readability a lot. But is this ordering mandatory? I mean even U++ has classes with members declared in "reverse" order.
In fact I used to declare class members in traditional order (public, protected, private) in the past, but changed my habit after I got used to U++ code base. :) Nevertheless, no problem.
it will only take several seconds to reverse the order. YEt, imho, this is more a matter of aesthetic taste.

Quote:
We can replace following code with the object approach

Well, yes. In fact, I initially designed it with objective approach. Bu then decided otherwise.
Not that it wasn't useful. I had simply felt that it was bloating the code (this was maybe a side effect of being a former C programmer :)).

But what you propose is reasonable, actually.
I can depreceate the current convenience functions in favour of an objective approach.
Given the flexibility of FTP class and package, it will be very easy. FtpAsyncPut() and FtpAsyncGet() functions call, under the hood, FtpAsyncIO which has one more parameter to determine the direction of transfer.
An Ftp::Request object can take care of both synchronous and asynchronous conveninence functions' parameter list, without giving any headache, acting as an intermediate layer.

FtpGet(Ftp::Request(local_file, remote_file, localhost).SSL().Passive(), progress, whenwait)

FtpAsyncGet(Ftp::Request(local_file, remote_file, localhost).SSL().Active(), progress)

Also, I can change the Event<int, int, String, int64, int64>, which, frankly, I am not happy with, into an Event<Ftp::Result> (which can be returned by also (FtpGet and FtpPut):

```
Ftp:: Result {

public:
	int    GetErrorCode() const;
	String GetErrorDesc() const;
	int64  GetTotal() const;
	int64  GetDone() const;
	// Async only.
	int    GetId() const;

	bool   IsSuccess() const;
	bool   IsFailed() const;
	// Async only.
	bool   InProgress() const;

	Result() {}

private:
// members...
};

//
// If we return Ftp::Result from FtpGet()

 if(FtpGet(...).IsSuccess())
   //....
 else
   //....

 // OR

  Event<Ftp::Result> r;
  FtpAsyncGet(..., r, ...);

  // In somewhere else, probably in the main thread:

void Foo::TransferProgress(Ftp::Result r) {

// First take care of UI thread synchronization, using whatever necessary, e.g PostCallback.
// then simply...

   if(r.InProgress())
```

```
    some_ctrl.Text(r.GetTotal(), r.GetDone());
  else
  if(r.IsSuccess())
    some_ctrl.Text("Done.");
  else
    some_ctrl.Text("Failed".);

}
```

What do you think?

Regards,

Oblivion