Subject: Re: SFTP or full SSH2 support for U++?
Posted by Oblivion on Tue, 27 Jun 2017 23:13:27 GMT
View Forum Message <> Reply to Message

Hello everyone,

It is now time to publish the technical preview version of SSH package for Upp.

A brief status of the project:

SSH package for U++

SSH package is a libssh2 wrapper for Ultimate++.
It aims to be a complete secure shell client solution. (SFtp, Scp, Exec, Terminal, X11Exec, KnownHosts)
Currently it is at a technical preview stage, which means it is a work-in-progress, and subject to change.

Requirements:

- A C++ compiler with at least C++11 support.
- libssh2 ssh library (https://www.libssh2.org/)

Features and Highlights:

- Tight integration with U++.
- Support for both synchronous and asynchronous operations with both high level and low level methods.
- Allows using U++ style memory [de/re]allocators with libssh2.
- Ssh subsystems support RTTI, and share a common interface for polymorphism.
  Namely, it is possible to store different subsystems in the same U++ container.
  (e.g. SFtp, Exec, Scp, Shell can be all stored in the same Array.)

Todo:

- Add known hosts and ssh agents support.
- Add more high level methods.
- Add ssh X11 support.
- Ship libssh2 library with the package.
- Documentation.

History:

2017-06-28: EAGAIN is now properly handled across the subsystems.
        WhenWrite and WhenRead callbacks are removed in favour of parametrized gates.

2017-06-27: SFtp::StartStop: fixed.
2017-06-27: CleanUp() methods are added to the classes Ssh, SFtp, and Channel.
        This method utilizes JobQueue's WhenCleanup event.
2017-06-26: U++ memory [de/re]allocators added and made default.
        initial support for ssh shell, and terminal added.
        Subsystems now perform a clean up on failure to prevent possible heap leaks.
2017-06-22: Initial release.

At the moment I am documenting the Package and its api.
There are some important points you need to know before getting yourself familiar with the code.

As I've mentioned in the above summary, SSH package is a libssh2 wrapper. But currently it does not contain libssh2 source code (which also has BSD license).
On linux, your distribution probably provides the lib. On Windows you'll need cmake for express compilation.

Design:

The classes of SSH package are based on JobQueue class, a simple job-queue model (a callback queue based on Upp::Vector actually).
JobQueue is meant to work with (basically) sockets. It provides a uniform interface for async socket operations. In fact, it is an abstraction of Upp's own HttpRequest class' async interface. From my experience, I can say that it is pretty scalable with a negligable overhead.It allows batch processing (e.g. you can queue the commands and then batch-process them later.)
However if you are going to batch process some commands, do note that the queue will treat the batch as a whole. Hence a failure in one command will halt and clear the queue. This is a security measure to prevent any misbehaviour.

Other advantage of JobQueue is that it allows writing code in a uniform coding pattern. (You can see it in SSH package clearly), in the sense that it exposes both a small set of common functions, and a basic structure.
All SSH sub/classes are based on JobQueue. Therefore they expose a uniform asynchronous api.

Also, each blocking method has its nonblocking counterpart (starting with "Start" prefix.)

Categorization:

SSH package is divided into several classes, based on system-subsystems relationship.
[
Ssh   -> Main session class (system). Provides connection and authentication mechanism. Although known Hosts support not added yet, it will be available in the following versions.

SshSubsystem -> SshSubsystem is base of all SSH protocol based communication subsystems.

SFtp   -> Complete, but I'm planning to further enhance it with more high level commannds,

once the api stabilized.
Channel -> Mostly done.
Scp -> Complete.
Exec -> Complete.
Shell -> Not added yet. Besic support is present. And will be developed durther.
Terminal -> Not added yet. Basic support is present. And will be developed further.
X11Exec -> Not added yet. Will be added incrementally.

Core class, as one might expect is, Ssh.
It provides connection and authentication mechanisms.

All other Ssh subsystems are derived from a single base class: SshSubsystem.
This class provides common operations and connects subsystems to the Ssh session.

Also it has RTTI interface, which brings us to polymorphism:
SSH package allows a great level of polymorphism while keeping the crucial interface uniform throughout the derivative classes (thanks to JobQueue).
Hence it is possible to put, say, a Scp, Exec, Sftp, Terminal in a single Upp::Array and execute them asynchoronusly through a uniform calling pattern.

All other classes -except SFtp- are derived directly from Channel subsystem. All derivatives expose Channel commands, while has their own commands. So any Channel-based derivative use the power of Channel.

Memory Allocation:

libssh2 provides support for custom allocators. (or alternatively it can use system's memory allocators via USEMALLOC flag)
Basically I copied and adjusted the memory manager found in Core/SSL package.
It works fine with libssh2, but I'm not very happy with it (code duplication is something I don't like)
Maybe there is a better way?

How To:

Package currently contains three simple examples:

SshExec: This simple example demonstrates a remote command execution via ssh2 protocol's exec channel, and the blocking api of SSH package.
SFtpDir: This simple example demonstrates a remote diretory listing via sftp channel, and the blocking api of SSH package.
SshAsyncRequests: This simple example demonstrates polymorphism with ssh subsystems, and asynchronous calls, combining a SFtp and Exec instance.

In fact, the first two example are so simple that they look like little code snippets:

SshExec:

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

using namespace Upp;

// SshExec.cpp:
// Demonstrates remote command execution on an ssh channel.

// A popular public test server:
// ----------------------------
// Host:     test.rebex.net
// User:     user
// Password: password
// Command:  ls -l

CONSOLE_APP_MAIN
{
 Ssh session;
 Cout() << "Hostname: ";
 auto host = ReadStdIn();
 Cout() << "Username: ";
 auto user = ReadStdIn();
 Cout() << "Password: ";
 auto pass = ReadStdIn();
 const int port = 22;

 if(session.Connect(host, port, user, pass)) {
  Cout() << "Command: ";
  auto cmd = ReadStdIn();
  Exec xc(session);
  auto rc = xc(cmd, Cout(), Cerr());
  if(xc.IsFailure()) Cerr() << Format("Exec failed. %s", xc.GetErrorDesc());
  else Cout() << Format("Remote program exited with code: %d\n", rc);
 }
 else Cerr() << session.GetErrorDesc() << '\n';
}
```

Or, SFtpDir:

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

using namespace Upp;

// SFtpDir.cpp:
```

```
// Demonstrates remote directory listing via an sftp channel.

// A popular public test server:
// ----------------------------
// Host:     test.rebex.net
// User:     user
// Password: password
// Path:     pub/examples

CONSOLE_APP_MAIN
{
 Ssh session;
 Cout() << "Hostname: ";
 auto host = ReadStdIn();
 Cout() << "Username: ";
 auto user = ReadStdIn();
 Cout() << "Password: ";
 auto pass = ReadStdIn();
 Cout() << "Path: ";
 auto path = ReadStdIn();
 const int port = 22;

 if(session.Connect(host, port, user, pass)) {
  SFtp sftp(session);
  SFtp::DirList ls;
  if(!sftp.ListDir(path, ls)) Exit(sftp.GetError());
  for(auto& e : ls)  Cout() << e.GetEntry() << "\n";
 }
 else Cerr() << session.GetErrorDesc() << '\n';
}
```

That's it

Please keep in mind that while ths package is reasonably stable it is still a work-in-progress. So, USE IT AT YOUR OWN RISK.

Although, given my personal experience I am pretty satisfied by SSH package's api, it is by no means set in stone. It'll change when necessary.
I'll be grateful for any criticism, suggestions, reviews, patches, etc. After all, this package is for U++ and its users.

Package tested on Win7/10 (Mingw64), and latest Arch Linux and Fedora (25).

p.s. There is a leftover that must be changed, which I realized this morning. (In SshSubsystem::To() method) I will change it tonight, but in the meanwhile if you are going to test the package please consider changing the reinterpret_cast to dynamic_cast.

Cheers,
Oblivion.

File Attachments

1) SSH (Technical Preview and Examples).zip, downloaded 440 times