**QGempis**

# Introduction

QGempis is a C++ program that sends Gempis commands directly from Windows to a Unix cluster node. QGempis relies on Qt and the Chilkat (http://www.chilkatsoft.com) SSH library. There are **no** Cygwin dependencies.

# Legal aspect

One has to be aware that the SSH library is probably subject to export control rules. By linking this library to our program, some legal constraints could complicate its deployment .

A possible alternative is to get rid of the library, and instead use plink which is a separate executable. This solution will impact the class SSHSession (see below), more code will be necessary to parse the output text from the external executable. The rest of the code should remain unchanged.

# Description

QGempis uses Qt for the GUI and multi-threaded mechanisms. Although the GUI is quite rudimentary, it has basic functionalities:

* Password settings
* Job launching

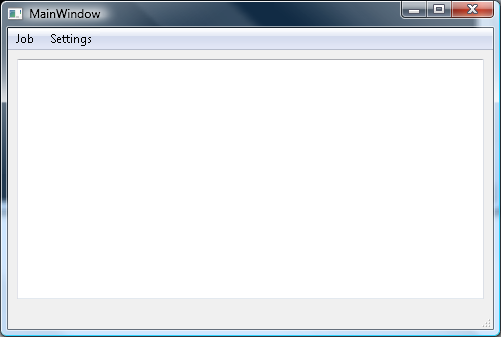
QGempis has 2 modes:

* With user interface
* command line mode

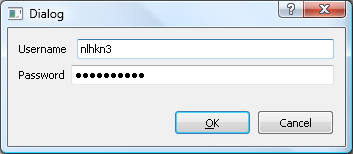
If no arguments are given, QGempis will launch with a GUI. In command line mode, QGempis will send the first argument to the remote server (for example: qgempis "ls -lsa").

The figure below shows the main screen of the application. The main area of the application will show jobs results.

The user can set his username and password by clicking on Settings.

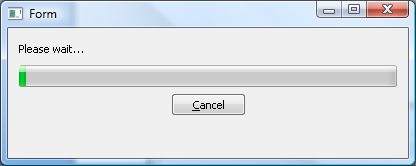


The password window is shown below.



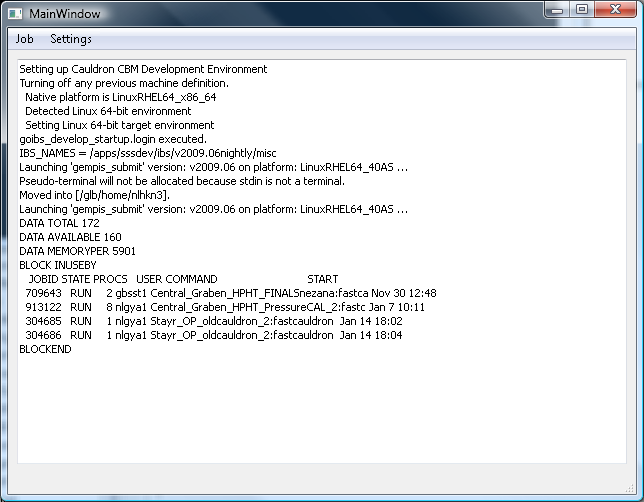
Username and passwords are stored unscrambled in the registry. Each time a connection is made, the username and password will be passed. Note that it is possible to use a public/private key mechanism, but it would probably increase the program complexity.

By clicking on Job, a connection is made to the cluster head node, a job is launched and the window below appears.



The progress bar will advance to show that the program is running. It is possible at any point to cancel the transaction.

At this stage, the job is hard coded and displays the resources on the Darwin2 cluster (see figure below for a result example).



## Classes

Besides the usual GUI classes (in our case MainWindow), the main classes are CkSsh, SSHSession and BackgroundSSHSession.

At the lowest level, CkSsh from Chilkat (http://www.chilkatsoft.com) is used.

CkSsh is encapsulated within the class SSHSession.

### SSHSession

As mentioned before, this class encapsulates the Chilkat one. The 2 main functions are Connect() and ExecuteCommand(). As the name suggests, Connects() creates a connection to remote computer (hardcoded for now, see file MyDefines.h). ExecuteCommand() sends a command to the remote computer, the output of the command is stored in a string.

This classes executes in the foreground, this means that this class cannot be called directly from a GUI event loop, or the application will freeze until the SSH transaction is completed. Note that this class is used directly in the command line version of QGempis.

As mentioned earlier, it should be possible to replace the Chilkat library with the plink executable. Dynamo already plink for the SSH communication. To have a comparable implementation, the SSHSession class needs rewriting. The code will be extended with process forking and output parsing. It is estimated that 1 week should suffice to replace SSH Chilkat with plink.

In Dynamo, those functions are located in the files Process.cpp and RemoteRunner.cpp.

### BackgroundSSHSession

The BackgroundSSHSession allows the program to make a connection, but not freeze the GUI. This makes possible for the user to start operations while a connection is in progress (such as cancel).

When running, a background SSH session will create a thread, instantiate a SSHSession object and use it inside the new thread.

## Scripts

On the remote server, the gempis script is called from a small wrapper. This wrapper (see below) is necessary to set paths to find the bjobs executable.

#!/bin/csh

source /nfs/rvl/users/ibs/bin/gocbm3

/apps/sss/share/gempis $\*

## Plink

The replacement of the SSH library by Plink takes place in the class SSHSession. The most difficult aspect is the communication with Plink. To communicate with plink, pipes need to be created and parsed, for example to catch error messages because of a wrong password.

The dynamo code uses anonymous pipes. With pipes it is difficult to check whether all the output has been read, the trick that dynamo uses is to continue reading until a specific string has been found.

As 2 ways communication is not implemented yet (no input is sent to the remote computer), it assumed that the finger-print of the remote computer is already known, if not then execute plink outside of the prototype and make a connection to the remote computer.