Multi-user Remote Test Harness:
The Remote Test Harness is a project that consists of three entities: the Client, the Test Harness Core and the Child processes. The project mainly deals with running a set of test libraries selected by the client in a sequential manner by the Child process. These are scheduled by the Remote Test Harness Core. The results are then logged back to the output stream, a persistent storage and a copy is given back to the client. Multiple Test Libraries are developed each of which consist of multiple tests which can be executed sequentially. Tests can be added to any of thee libraries without disturbing any other function execution. Each test follows a common interface. The exception handling is taken care of for each tests so even if a test fails, it does not stop the later tests of the library to execute and just returns failure for the current failed test. A Test Request is sent by the Client to the Test Harness core that contains the Test Library converted to the DLL format implementing all the tests, reference to the code that is to be tested and the logger which logs the results to the client. The Test Harness core contains two queues: Ready Queue and the Test Request Queue. Each Test request from the client is stored in the Test Request queue until it goes to one of the Child process for execution. The Ready Queue saves the state of the Child processes and whenever one is ready, it enqueues to the Ready Queue. When a test request is received by the core, it tries to dequeue the a ready message and until and unless it finds one, it is blocked by the Test Request queue (which explains the thread safe execution of blocking queues). If a ready message is found in the Ready queue, then the Test Request is sent to the child that sent the ready signal. The child process then loads the Test library and executes all the tests sequentially in that particular library. When this test is complete, the test logs are sent back to the client. This whole communication between Client, Harness core and Child process takes place with the help of a Message Passing Communication channel. This communication framework address the message for requests, replies and notifications. The client can select the test libraries by an interactive GUI built using .NET Windows Presentation Foundation(WPF), which uses C#. Because of this, a shim converter layer needed to be developed for the effective execution of the C# request into native C++ implementation. This layer was built using C++/CLI.