Versão Java

A versão Java do cliente foi testada utilizando o JDK 5.0 e 6.0, porém como não implementa nenhuma função adicionada recentemente à API, muito provavelmente irá funcionar sem problemas até a versão 1.3 do JDK.

Basicamente, são duas classes que são necessárias para a utilização do cliente, a **QuanserClient** e a **QuanserClientException**. A Classe `QuanserClient` é o cliente propriamente dito, onde temos como construtor único `QuanserClient(String _server, int _tcpPort)`, onde o parâmetro `_server`, é o servidor onde se encontra a placa AD/DA, e `_tcpPort`, é a porta de escuta desse servidor. Atualmente, na versão atual do servidor (0.1.98/Windows/Borland C++ Builder 6.0) a porta de escuta padrão é a `20072`. E a `QuanserClientException`, nada mais é que a classe de tratamento de exceções padrão.

Essas classes são disponibilizadas através do arquivo **`QuanserClient.jar`**, que deve ser adicionado ao projeto, caso seja utilizado algum ambiente de desenvolvimento (ex.: Eclipse, Netbeans etc), ou informado em tempo de compilação e execução.

Os dois métodos que serão utilizados são o `read(int _channel)`, utilizado para Leitura no canal especificado em `_channel`. Nesse caso, o retorno é um Double, com o valor em volts no canal, e o `write(int _channel, double _volts)`, que escreve o especificado em `_volts`, no canal `_channel`.

Exemplo:

```
public static void main(String[] args) {
        QuanserClient quanserClient;
        try {
            quanserClient = new QuanserClient("10.13.97.69", 20072);
            double _read = quanserClient.read(0);
            System.out.println("Leitura Canal 0: " + _read);
            System.out.println("Gravando 1.2 volts no canal 4...");
            quanserClient.write(4,1.2);
        }
        catch (QuanserClientException ex) {
            ex.printStackTrace();
        }
}
```

Versão C++ (Linux /GCC)

A versão C++ em Linux foi testada somente no GCC 4.1, porém, por não implementar nada de excepcional, provavelmente funcionará sem problemas em versões anteriores do GCC.

Aqui, a utilização é ainda mais simples. Nessa primeira versão, erros não serão tratados como exceptions como na versão do Java, e sim como valores de retornos das funções de gravação e leitura dos canais.

Assim como na versão Java, instanciamos uma classe, porém o nome aqui é somente “Quanser”, através do include do arquivo `quanser.h`.

A utilização é muito semelhante a versão Java, e o exemplo abaixo é auto-explicativo.

Exemplo:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <iostream>
#include <string.h>

#include "quanser.h"

int main(int argc, char** argv) {
    Quanser* q = new Quanser("10.13.97.69", 20072);

    q->writeDA(4,1);
    double read = 0;
    read = q->readAD(4);
    cout << read << "\n";

    return (EXIT_SUCCESS);
}
```

Da mesma forma que no Java, temos o construtor que instancia e conecta ao servidor, e os métodos `writeDA` e `readAD`, que funcionam de forma análoga a versão em Java.

Versão C++ (Borland C++ Builder 2007)

Essa versão de cliente foi testada somente no Borland C++ Builder 2007, porém é muito provável que ela **NÃO** funcione de forma correta em versões anteriores (especialmente na versão 6.0). Alguns testes nos componentes de conexão TCP da versão 6.0 trouxeram resultados frustrantes, problemas estes que até o momento não ocorreram na versão 2007.

O construtor e métodos públicos são exatamente os mesmos da versão GCC/Linux, dispensando maiores explicações, o exemplo sendo auto-explicativo o suficiente.

Exemplo:

```
#include <vcl.h>
#include <Sockets.hpp>
#include <Classes.hpp>
#include <iostream.h>

#include "quanser.h"

#pragma argsused
int main(int argc, char* argv[])
{
      Quanser* quanser = new Quanser("10.13.97.69",20072);

      for (int i = 0; i < 10; i++) {
            double _received = quanser->readAD(0);
            std::cout << "received: " << _received << "\n";
            int _result = quanser->writeDA(1,1.11);
            std::cout << "result: " << _result << "\n";
      }

      Sleep(4000);
      return 0;
}
```

Obs.: Outras linguagens

Em virtude da simplicidade do protocolo de comunicação, a criação de clientes em outras linguagens se torna razoavelmente simples, demandando um breve estudo na linguagem, principalmente no tratamento de strings e conexões TCP/IP. Caso deseje utilizar outra linguagem, ou ainda portar alguma dessas versões de clientes para alguma linguagem por conta própria, sinta-se a vontade, só peço-lhe para contribuir e informar, para que outras pessoas também usufruam.

É objeto desse trabalho, criar clientes ainda para *Python* e *C#,* bem como adicionar funcionalidades a alguns dos clientes até o final do semestre.