F. Wagner February 2004

How to Write a GUI for StateWORKS Applications

Graphical User Interface for StateWORKS Run-time Systems

Most applications need some sort of graphical user interface (GUI). The GUI is used to communicate with the program: to set parameters, send commands, display results, show alarms, etc. In a StateWORKS run-time system the entire information is stored in the Real Time Data Base (RTDB). RTDB has a TCP/IP interface which allows access to the data. The RTDB operates as a server. To access the data in the RTDB we need a TCP/IP client.

Monitoring programs that are part of the StateWORKS Studio are programs that use a TCP/IP interface for communication with the RTDB: they are RTDB clients.

To make the task easy we provide a TCP/IP Client library which should be used for building any GUI interface for a StateWORKS application.

TCP/IP Client Library

The TCP/IP C++ library has been built and tested with MS Visual Studio. To use it 3 h-files are required. The most interesting h-file TCPIPClient.h contains the declaration of all functions used for building GUI programs:

-	Initialize()	to store the callback function and environment pointer,
-	Connect()	to create socket and the RTDB server,
-	Connected()	to test whether the client is connected,
-	Disconnect()	to disconnect the client from the RTDB server
-	Request()	to get a value of the object property, the actual value is returned as a function parameter and via the event port
-	Poke()	to set a value of the object property, the set value is returned as a function parameter
-	AdviseStart()	to start advise of the object property
-	AdviseStop()	to stop advise of the object property
-	UnAdviseAll()	to stop advise

The port created with the TCP/IP library contains two sockets: one for sending Request, Poke, etc. and the second one for events generated by Advised objects or in reply to Request.

You can learn from the GUI_CPP.cpp file (Appendix 1) the details of the connection. The file has been taken from the GUI_CPP.exe project prepared with MS Visual Studio .NET. Of course, the details of windows programming will differ for other development platform. Therefore I ignore them and concentrate only on TCP/IP relevant topic. The description below contains some detailed information explaining the idiosyncrasies of the library functions.

The initialization

The `Initialize()` method called by a start of the program in the `OnInitDialog()` requires two parameters:

§ the reference to the callback function RepEvt used by the event thread run in the TCP/IP client.

§ the pointer to the *Owner* of the RepEvt function.

The Connection

The `Connect()` method called in the `OnBnClickedConnect()` requires two parameters: the *server name* and the *port number*. As both parameters have default values: respectively LOCALHOST and 9091 (the StateWORKS port number) they can be omitted if the client will run on the same computer as the StateWORKS run-time system.

The Disconnection

The `Disconnect()` method called in a local `Disconnect()` function does not take any parameters (actually, it has one parameter *bWithUnadvise* that you can ignore as it has a default value *false*). The local `Disconnect()` function is used twice: in `OnBnClickedDisconnect()` activated when the user disconnects the client with the Disconnect button and in the RepEvt function when the server terminates the connection.

Getting a value

The `Request()` method is called several times: by Connect, Get, Set and in the RepEvt function. It should be called by setting a value in the RTDB as the automatic answer coming to the Write value operation is not the actual value but the written value.

The Request() method requires 3 parameters:

§ the *object Name*, for instance *"Ni:ActualPressureValue",*

§ the *object Attribute* to be read (see Appendix), for instance *IATT_Va*l,

§ the reference to a *Value* variable which will contain the read attribute value.

Setting a value

The `Poke()` method is called in the `OnBnClickedSet()` function and requires 3 parameters:

§ the *object Name*, for instance *"Ni:ActualPressureValue",*

§ the *Value* to be written, for instance *IAtt_Trc*,

§ the *object Attribute* to be written (see Appendix), for instance *.Va*l.

As mentioned already before, it makes sense to complete the writing an attribute value by a reading operation because the value returned during the write operation is not the actual value but the written value (a kind of echo).

Advise an object attribute

The Advise operation instructs the server to send an event to the client if the "advised" object attribute has changed. The `AdviseStart()` method is called in the `OnBnClickedAdvise()` function and requires 3 parameters:

§ the *object Name*, for instance *"Ni:ActualPressureValue",*

§ the *object Attribute* to be advised (see Appendix), for instance *IATT_Va*l,

§ the reference to *a Value* variable which will contain the just advised attribute value (for an advise operation the actual value is returned).

Unadvising an object attribute

The Unadvise operation instructs the server to stop sending events to the client. The `AdviseStop()` method is called in the `OnBnClickedUnAdvise()` function and requires 2 parameters:

- the *object Name*, for instance *"Ni:ActualPressureValue",*
- the *object Attribute* to be unadvised (see Appendix), for instance *IATT_Va*l.

Note that all methods parameters except the Name of the object and Attribute in case of the Poke() method has default values. If you do not set the Attribute value the functions return doing nothing.

The RepEvt function

The `RepEvt()` function cannot belong to the Application class (in this case CGUI_CPPDlg). Therefore while called it receives the pointer to the environment which allows it to use the method of the application class.

The `RepEvt()` function is called in the client event thread implemented in the TCP/IP library (not accessible to the programmer). It is called there in 3 relevant cases:

- when the server send an event with the advised data object,
- when the server restarts the application configuration file,
- when the server exits.

It is called also by request but normally we do not use it as we get the requested value anyway when the `Request()` function returns.

Example GUI_CPP

The GUI interface depends on the application. It is impossible to write something of a general character, except Monitors which offer access to all RTDB data, so the example given is a kind of Monitor. The source code, which is provided for download from our web site, shows the usage of the TCP/IP Client functions. In addition to its tutorial function, the example displays all object attributes and allows quickly to check the read/write feature of the attribute – clicking the get / set buttons produces a message if the operation is not allowed.

Running the example

The GUI_CPP.exe example is included in the StateWORKS Studio package. You find it in the Installation directory.

While running any StateWORKS application, for instance SWLab, start the SWQuick Monitor program which uses the described GUI interface. The usage is so obvious that it does not make sense to provide any Help or description.

Summary

With the TCP/IP Client Library the user gets an important component for linking the Graphical User Interface with StateWORKS run-time systems. This library can also be used to build any other kinds of interfaces to communicate with the run-time system (an example will be available on our web site soon).

Appendix 1

```
// GUI_CPPDlg.cpp : implementation file
//

#include "stdafx.h"
#include "GUI_CPP.h"
#include "GUI_CPPDlg.h"
#include ".\gui_cppdlg.h"
#include <algorithm>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
      CAboutDlg();

// Dialog Data
      enum { IDD = IDD_ABOUTBOX };

      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support

// Implementation
protected:
      DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()


// CGUI_CPPDlg dialog



CGUI_CPPDlg::CGUI_CPPDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CGUI_CPPDlg::IDD, pParent)
{
      m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CGUI_CPPDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CGUI_CPPDlg, CDialog)
```

```
      ON_WM_SYSCOMMAND()
      ON_WM_DESTROY()
      ON_WM_PAINT()
      ON_WM_QUERYDRAGICON()
      //}}AFX_MSG_MAP
  ON_LBN_SELCHANGE(IDC_OBJECTLIST, OnLbnSelchangeObjectlist)
  ON_BN_CLICKED(IDC_CONNECT, OnBnClickedConnect)
  ON_BN_CLICKED(IDC_DISCONNECT, OnBnClickedDisconnect)
  ON_BN_CLICKED(IDC_GET, OnBnClickedGet)
  ON_BN_CLICKED(IDC_SET, OnBnClickedSet)
  ON_BN_CLICKED(IDC_ADVISE, OnBnClickedAdvise)
  ON_BN_CLICKED(IDC_UNADVISE, OnBnClickedUnadvise)
END_MESSAGE_MAP()


// CGUI_CPPDlg message handlers

BOOL CGUI_CPPDlg::OnInitDialog()
{
      CDialog::OnInitDialog();

      // Add "About..." menu item to system menu.

      // IDM_ABOUTBOX must be in the system command range.
      ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
      ASSERT(IDM_ABOUTBOX < 0xF000);

      CMenu* pSysMenu = GetSystemMenu(FALSE);
      if (pSysMenu != NULL)
      {
            CString strAboutMenu;
            strAboutMenu.LoadString(IDS_ABOUTBOX);
            if (!strAboutMenu.IsEmpty())
            {
                  pSysMenu->AppendMenu(MF_SEPARATOR);
                  pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
            }
      }

      // Set the icon for this dialog.  The framework does this
automatically
      //  when the application's main window is not a dialog
      SetIcon(m_hIcon, TRUE);            // Set big icon
      SetIcon(m_hIcon, FALSE);           // Set small icon

      // Application initialisations

  // Pass the callback function to TCPIPClient
  m_Client.Initialize(RepEvt, this);

  CEdit *pHost = (CEdit*)GetDlgItem(IDC_HOST);
  pHost->SetWindowText( HOST );

  CEdit *pPort = (CEdit*)GetDlgItem(IDC_PORT);
  char stPortNumber[16];
  itoa(PORTNUMBER, stPortNumber, 10);
  pPort->SetWindowText( stPortNumber );

  CEdit *pResult = (CEdit*)GetDlgItem(IDC_RESULT);
  pResult->SetWindowText( "Disconnected" );

      return TRUE;  // return TRUE  unless you set the focus to a control
```

```
}

void CGUI_CPPDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
      if ((nID & 0xFFF0) == IDM_ABOUTBOX)
      {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
      }
      else
      {
            CDialog::OnSysCommand(nID, lParam);
      }
}

void CGUI_CPPDlg::OnDestroy()
{
      WinHelp(0L, HELP_QUIT);
      CDialog::OnDestroy();
}

// If you add a minimize button to your dialog, you will need the code
below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CGUI_CPPDlg::OnPaint()
{
      if (IsIconic())
      {
            CPaintDC dc(this); // device context for painting

            SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

            // Center icon in client rectangle
            int cxIcon = GetSystemMetrics(SM_CXICON);
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;

            // Draw the icon
            dc.DrawIcon(x, y, m_hIcon);
      }
      else
      {
            CDialog::OnPaint();
      }
}

// The system calls this function to obtain the cursor to display while the
user drags
//  the minimized window.
HCURSOR CGUI_CPPDlg::OnQueryDragIcon()
{
      return static_cast<HCURSOR>(m_hIcon);
}

void CGUI_CPPDlg::OnLbnSelchangeObjectlist()
{
  CListBox *pObjectList = (CListBox*)GetDlgItem(IDC_OBJECTLIST);
  CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
  CListBox *pReceivedList = (CListBox*)GetDlgItem(IDC_RECEIVED);

  // Get selected entry in object list
  int iIndex = pObjectList->GetCurSel();
  CString stItem;
  pObjectList->GetText(iIndex, stItem);

  // Get item type
  int i = stItem.Find(':');
  stItem = stItem.Left(i);  // get item type (string)
  for (i=(int)IT_Item; i<(int)IT_last; i++)
    if (stItem == aszItemTypes[i])
      break;
  e_ItemTypes eType = (e_ItemTypes)i;

  // Copy appriociate attributes to the Attribute list
  // and initilaize the Received list
  pAttrList->ResetContent();
  pReceivedList->ResetContent();
  for (i=(int)IAtt_None+1; i<(int)IAtt_last; i++)
  {
    if (abAttribute[i][eType] != a)
    {
      pAttrList->AddString(aszItemAttName[i]);
      pReceivedList->AddString("");
    }
  }
  pAttrList->AddString(". (all)");
}

void CGUI_CPPDlg::OnBnClickedConnect()
{
  CString stHost;
  CEdit *pHost = (CEdit*)GetDlgItem(IDC_HOST);
  pHost->GetWindowText( stHost );
  string stHost1 = stHost;

  if ( m_Client.Connected() )
    MessageBox( "Already connected" );
  else
  {
    if ( m_Client.Connect(stHost1) )
    {
      CEdit *pResult = (CEdit*)GetDlgItem(IDC_RESULT);
      pResult->SetWindowText( "Connected" );
      // Get all object names
      string stType;
      string stValue;
      int    iIndex;
      string stItem;
      CListBox *pObjectList = (CListBox*)GetDlgItem(IDC_OBJECTLIST);
      for (int i=IT_VFSM; i<IT_last; i++)
      {
        stType = aszItemTypes[i];
        if ( m_Client.Request( stType, IAtt_List, stValue ) )
        {
          if ( !stValue.empty() )
          {
            // one line contains all objects of a given type separated by
LF
            // add the object one by one into the list;
            // put the object type in front of the object name
            iIndex = stValue.find('\n');
            while ( iIndex > 0 )
            {
              stItem = stType + ": " + stValue.substr(0, iIndex);
              pObjectList->AddString( stItem.c_str() );
              stValue = stValue.substr( iIndex+1, stValue.length() );
              iIndex = stValue.find('\n');
            }
          }
        }
      }
    }
    else
    {
      MessageBox( "Cannot connect to Server" );
    }
  }
}

void CGUI_CPPDlg::OnBnClickedDisconnect()
{
  if ( !m_Client.Connected() )
    MessageBox( "Already disconnected" );
  else
    Disconnect();
}


void CGUI_CPPDlg::OnBnClickedGet()
{
  if (!m_Client.Connected())
  {
    MessageBox("Not connected");
    return;
  }

  string stItemName;
  CString stItemType;
  if (!GetItem(stItemType, stItemName))
    return;

  string stAttr;
  int    iIndex;
  if (!GetAttribute(iIndex, stAttr))
    return;

  CListBox *pReceivedList = (CListBox*)GetDlgItem(IDC_RECEIVED);
  CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
  string stValue;

  e_ItemAttributes eAttr;
  if (iIndex == 0) // . (all) attribute selected
  {
    for (iIndex=1; iIndex<pAttrList->GetCount(); iIndex++)
    {
      FindAttribute(MT_Request, iIndex, stItemType, eAttr);
      if (m_Client.Request(stItemName, eAttr, stValue))
      {
        pReceivedList->DeleteString(iIndex);
        pReceivedList->InsertString(iIndex, stValue.c_str());
      }
    }
  }
  else
  {
    FindAttribute(MT_Request, iIndex, stItemType, eAttr);
    if (m_Client.Request(stItemName, eAttr, stValue))
    {
      pReceivedList->DeleteString(iIndex);
      pReceivedList->InsertString(iIndex, stValue.c_str());
    }
  }
}


void CGUI_CPPDlg::OnBnClickedSet()
{
  if ( !m_Client.Connected() )
  {
    MessageBox("Not connected");
    return;
  }

  string stItemName;
  CString stItemType;
  if (!GetItem(stItemType, stItemName))
    return;

  string stAttr;
  int    iIndex;
  if (!GetAttribute(iIndex, stAttr))
    return;

  if (stAttr == ". (all)")
  {
    MessageBox("Cannot set all attributes. Select one only");
    return;
  }
  e_ItemAttributes eAttr;
  FindAttribute(MT_Poke, iIndex, stItemType, eAttr);
  if (eAttr == IAtt_None)
    return;

  CString stValue1;
  string stValue;
  CEdit *pToSent = (CEdit*)GetDlgItem(IDC_TOSENT);
  pToSent->GetWindowText( stValue1 );
  if (stValue1 == "")
  {
    int iResult;
    iResult = MessageBox("Would you really like to write an empty string
(0)?", NULL, MB_YESNO);
    if (iResult == IDNO)
      return;
  }
  stValue = stValue1;

  CListBox *pReceivedList = (CListBox*)GetDlgItem(IDC_RECEIVED);
  if (m_Client.Poke(stItemName, stValue, eAttr))
  {
    // The value returned by Poke is not the value really written
    // Therefore we read it
    e_ItemTypes eType;
    for (int i=IT_Item; i<IT_last; i++)
    {
      if ( stItemType.Compare(aszItemTypes[i]) == 0 )
```

```
        eType = (e_ItemTypes)i;
      }
      if (abAttribute[eAttr][eType] != aW)
      {
        if (m_Client.Request(stItemName, eAttr, stValue))
        {
          pReceivedList->DeleteString(iIndex);
          pReceivedList->InsertString(iIndex, stValue.c_str());
        }
      }
    }
    else
      MessageBox("Cannot set");
}


void CGUI_CPPDlg::OnBnClickedAdvise()
{
  if ( !m_Client.Connected() )
  {
    MessageBox("Not connected");
    return;
  }

  string stItemName;
  CString stItemType;
  if (!GetItem(stItemType, stItemName))
    return;

  string stAttr;
  int    iIndex;
  if (!GetAttribute(iIndex, stAttr))
    return;

  string stValue;
  e_ItemAttributes eAttr;
  if (stAttr == ". (all)")
  {
    CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
    CString stAttr1;
    for (iIndex=1; iIndex<pAttrList->GetCount(); iIndex++)
    {
      pAttrList->GetText(iIndex, stAttr1);
      FindAttribute(MT_AdvStart, iIndex, stItemType, eAttr);
      m_Client.AdviseStart(stItemName, eAttr, stValue);
    }
  }
  else
  {
    FindAttribute(MT_AdvStart, iIndex, stItemType, eAttr);
    m_Client.AdviseStart(stItemName, eAttr, stValue);
  }
}


void CGUI_CPPDlg::OnBnClickedUnadvise()
{
  if ( !m_Client.Connected() )
  {
    MessageBox("Not connected");
    return;
  }
  string stItemName;
  CString stItemType;
  if (!GetItem(stItemType, stItemName))
    return;

  string stAttr;
  int    iIndex;
  if (!GetAttribute(iIndex, stAttr))
    return;

  string stValue;
  e_ItemAttributes eAttr;
  if (stAttr == ". (all)")
  {
    CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
    CString stAttr1;
    for (iIndex=1; iIndex<pAttrList->GetCount(); iIndex++)
    {
      pAttrList->GetText(iIndex, stAttr1);
      FindAttribute(MT_AdvStart, iIndex, stItemType, eAttr);
      m_Client.AdviseStop(stItemName, eAttr);
    }
  }
  else
  {
    FindAttribute(MT_AdvStart, iIndex, stItemType, eAttr);
    m_Client.AdviseStop(stItemName, eAttr);
  }
}


bool CGUI_CPPDlg::GetItem(CString& stItemType, string& stItemName)
{
  // Get Item
  CListBox *pObjectList = (CListBox*)GetDlgItem(IDC_OBJECTLIST);
  int iIndex = pObjectList->GetCurSel();
  if (iIndex < 0)
  {
    MessageBox("Item not selected");
    return false;
  }
  CString stItem;
  pObjectList->GetText(iIndex, stItem);

  // Partition Item into type and name
  iIndex = stItem.Find(':');
  stItemType = stItem.Left(iIndex);
  stItemName = stItem.Right(stItem.GetLength() - iIndex - 2);
  return true;
}


bool CGUI_CPPDlg::GetAttribute(int& iIndex, string& stAttr)
{
  CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
  iIndex = pAttrList->GetCurSel();
  if (iIndex < 0)
  {
    MessageBox("Attribute not selected");
    return false;
  }
  CString stAttr1;
  pAttrList->GetText(iIndex, stAttr1);
  stAttr = stAttr1;
  return true;
}


void CGUI_CPPDlg::FindAttribute(const e_MessageType& eMsgType, const int&
iIndex,
                                const CString& stItemType,
e_ItemAttributes& eAttr)
{
  CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
  CString stAttr;
  pAttrList->GetText(iIndex, stAttr);
  e_ItemTypes eType;
  for (int i=IT_Item; i<IT_last; i++)
  {
    if ( stItemType.Compare(aszItemTypes[i]) == 0 )
      eType = (e_ItemTypes)i;
  }

  // Find attribute
  eAttr = IAtt_None;
  for (int i=IAtt_None; i<IAtt_last; i++)
  {
    if ( stAttr.Compare(aszItemAttName[i]) == 0 )
    {
      eAttr = (e_ItemAttributes)i;
      switch (eMsgType)
      {
        case MT_Request:
          if (abAttribute[eAttr][eType] == aW)
          {
            CString stText;
            stText.Format("The %s property cannot be read",
aszItemAttName[eAttr]);
            MessageBox(stText);
            eAttr = IAtt_None;
          }
          break;
        case MT_Poke:
          if (abAttribute[eAttr][eType] == aR)
          {
            CString stText;
            stText.Format("The %s property cannot be set",
aszItemAttName[eAttr]);
            MessageBox(stText);
            eAttr = IAtt_None;
          }
          break;
        default:
          break;
      }
      break;
    }
  }
}


bool CGUI_CPPDlg::EventFromServer(CString stItem, CString stValue)
{
  // Partition into name and attribute
  int iIndex = stItem.Find('.');
  CString stItemName = stItem.Left(iIndex);
  string  stAttr = stItem.Right(stItem.GetLength() - iIndex);

  // Is the item selected?
  string  stItemNameSelected;
  CString stItemType;
  if (!GetItem(stItemType, stItemNameSelected))
    return false;
  if (stItemName != stItemNameSelected.c_str())
    return false;

  // Find attribute in the AttrList
  CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
  CString stAttr1;
  for (iIndex=1; iIndex<pAttrList->GetCount(); iIndex++)
  {
    pAttrList->GetText(iIndex, stAttr1);
    if (stAttr1 == stAttr.c_str())
      break;
  }

  // Display value
  if (iIndex < pAttrList->GetCount())
  {
    CListBox *pReceivedList =
(CListBox*)CGUI_CPPDlg::GetDlgItem(IDC_RECEIVED);
    pReceivedList->DeleteString(iIndex);
    pReceivedList->InsertString(iIndex, stValue);
  }

  return true;
}


void CGUI_CPPDlg::Restart()
{
  // clear lists
  CListBox *pObjectList = (CListBox*)GetDlgItem(IDC_OBJECTLIST);
  pObjectList->ResetContent();
  CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
  pAttrList->ResetContent();
  CListBox *pReceivedList = (CListBox*)GetDlgItem(IDC_RECEIVED);
  pReceivedList->ResetContent();
  CEdit *pToSent = (CEdit*)GetDlgItem(IDC_TOSENT);
  pToSent->SetWindowText("");

    // Get all object names
  string stType;
  string stValue;
  int    iIndex;
  string stItem;
  Sleep(100); // give the server enough time to reload the configuration
  for (int i=IT_VFSM; i<IT_last; i++)
  {
    stType = aszItemTypes[i];
    if ( m_Client.Request( stType, IAtt_List, stValue ) )
    {
      if ( !stValue.empty() )
      {
        // one line contains all objects of a given type separated by LF
        // add the object one by one into the list;
        // put the object type in front of the object name
        iIndex = stValue.find('\n');
        while ( iIndex > 0 )
        {
          stItem = stType + ": " + stValue.substr(0, iIndex);
          pObjectList->AddString( stItem.c_str() );
          stValue = stValue.substr( iIndex+1, stValue.length() );
          iIndex = stValue.find('\n');
        }
      }
    }
  }
  MessageBox("Restarted");
}


void CGUI_CPPDlg::Disconnect()
{
  m_Client.Disconnect();
  // clear lists
  CListBox *pObjectList = (CListBox*)GetDlgItem(IDC_OBJECTLIST);
  pObjectList->ResetContent();
  CListBox *pAttrList = (CListBox*)GetDlgItem(IDC_ATTR);
  pAttrList->ResetContent();
  CListBox *pReceivedList = (CListBox*)GetDlgItem(IDC_RECEIVED);
  pReceivedList->ResetContent();
  CEdit *pToSent = (CEdit*)GetDlgItem(IDC_TOSENT);
  pToSent->SetWindowText("");

  CEdit *pResult = (CEdit*)GetDlgItem(IDC_RESULT);
  pResult->SetWindowText( "Disconnected" );
}


void RepEvt(int Rep, const string& stName, const string& stVal, void*
pOwner)
{
CGUI_CPPDlg* pNode = (CGUI_CPPDlg*)pOwner;

  switch ( Rep )
  {
  case RT_AdvisedData:
    pNode->EventFromServer(stName.c_str(), stVal.c_str());
    break;
  case RT_RequestedData:
    break;
  case RT_Restart:
    pNode->Restart();
    break;
  case RT_Terminate:
    pNode->Disconnect();
    break;
  default:
    break;
  }
}
```

Appendix 2. Attributes

		CMD	VFSM	AL	TI	DO	DI	NI	NO	XDA	SWIP	OFUN	PAR	CNT	STR	UNIT	DAT	UDC	ECNT	TAB
	IAtt_None	M,	R,	R,	R,	R,	M,	M,	R,	R,	M,	R,	R,	R,	R,	R,	R,	R,	M,	M
.Val	IAtt_Value	M,	R,	R,	R,	R,	M,	M,	R,	R,	M,	R,	R,	R,	R,	R,	-,	R,	M,	M
.SvM	IAtt_ServiceMode	M,	M,	-,	-,	M,	M,	-,	-,	M,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.SvV	IAtt_ServiceValue	M,	M,	-,	-,	M,	M,	-,	-,	M,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.PeV	IAtt_PeripheralValue	M,	M,	-,	-,	R,	R,	-,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.VI	IAtt_VI	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.StN	IAtt_StateName	R,	-,	R,	R,	-,	-,	R,	R,	R,	-,	R,	-,	R,	R,	R,	-,	R,	R,	-
.AIL	IAtt_AssocItemList	R,	-,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	R,	-,	-,	-
.Typ	IAtt_TypeName	R,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	R,	-,	-,	-
.CnC	IAtt_CountConstant	-,	-,	M,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	M,	-,	-,	M,	-,	-
.CnR	IAtt_CountRegister	-,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	R,	-,	-,	R,	-,	-
.Cat	IAtt_Category	-,	-,	-,	R,	-,	-,	-,	-,	-,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-
.Frm	IAtt_Format	-,	-,	-,	-,	-,	-,	R,	R,	-,	-,	R,	-,	-,	-,	R,	-,	-,	R,	-
.Uni	IAtt_PhysicalUnit	-,	-,	R,	-,	-,	-,	R,	R,	-,	-,	R,	-,	-,	-,	R,	-,	-,	R,	-
.LiL	IAtt_LimitLow	-,	-,	-,	-,	-,	-,	-,	-,	M,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-
.LiH	IAtt_LimitHigh	-,	-,	-,	-,	-,	-,	-,	-,	M,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-
.IVa	IAtt_InitValue	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	R,	-,	M,	-,	-,	-,	-,	-,	-
.Dat	IAtt_DataValue	-,	-,	-,	-,	-,	-,	R,	R,	R,	-,	M,	-,	-,	-,	M,	-,	-,	M,	-
.Txt	IAtt_Text	-,	-,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.Ack	IAtt_Acknowledge	-,	-,	-,	W,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.Tim	IAtt_Time	-,	-,	-,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.ScF	IAtt_ScaleFactor	-,	-,	-,	-,	-,	-,	R,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.Ofs	IAtt_Offset	-,	-,	-,	-,	-,	-,	R,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.ScM	IAtt_ScaleMode	-,	-,	-,	-,	-,	-,	R,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.Lst	IAtt_List	R,	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.PAd	IAtt_PhysAddr	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	R,	-,	-,	-
.Com	IAtt_CommPort	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	R,	-,	-,	-
.Trc	IAtt_Trace	M,	M,	M,	M,	M,	M,	M,	M,	M,	-,	M,	-,	-,	M,	M,	-,	M,	M,	-
.RMo	IAtt_RunMode	M,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-
.NSt	IAtt_NextStep	R,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-,	-

-	- none
R	– read only
M	- read / write
W	– write only