

# ***XMLGUI* manual**

Version: 1.3

## Version history

<b>Date</b>	<b>Version</b>	<b>Author</b>	<b>Notes</b>
07-Sep-2005	1.0	Leszek Skorczynski	Initial version.
11-Sep-2005	1.1	Leszek Skorczynski	Added context help.
22-Sep-2005	1.2	Leszek Skorczynski	One screen shot replaced to show English texts.
20-Nov-2005	1.3	Leszek Skorczynski	Added chapter 6.4 about mixing release and debug versions.



<b>1. OVERVIEW .....</b>	<b>4</b>
<b>2. NECESSARY FILES.....</b>	<b>4</b>
<b>3. DEFINITION OF WINDOW XML FILES.....</b>	<b>4</b>
3.1. FILENAME.....	4
3.2. GENERAL SCHEMA .....	5
3.3. <WINDOW> OBJECT.....	5
3.3.1. <i>id attribute</i> .....	5
3.3.2. <i>initialview attribute</i> .....	5
3.3.3. <i>sizeable attribute</i> .....	6
3.4. <CONTROL> OBJECT.....	6
3.4.1. <i>id attribute</i> .....	6
3.4.2. <i>type attribute</i> .....	7
3.4.3. <i>choosefile attribute</i> .....	7
3.4.4. <i>filemask attribute</i> .....	7
3.4.5. <i>number attribute</i> .....	8
3.4.6. <i>dll attribute</i> .....	8
3.4.7. <i>function attribute</i> .....	8
3.4.8. <i>stop attribute</i> .....	9
3.4.9. <i>start attribute</i> .....	9
3.4.10. <i>show attribute</i> .....	9
3.4.11. <i>readonly attribute</i> .....	10
3.4.12. <i>output attribute</i> .....	10
3.4.13. <i>log attribute</i> .....	11
3.4.14. <i>helpid attribute</i> .....	11
3.4.15. <i>clear attribute</i> .....	11
3.4.16. <i>timeout attribute</i> .....	12
3.5. <ITEM> OBJECT.....	12
3.5.1. <i>id attribute</i> .....	12
<b>4. FUNCTIONALITY .....</b>	<b>13</b>
4.1. STARTUP SEQUENCE.....	13
4.2. CALLING AN EXTERNAL FUNCTION .....	13
4.3. EXTERNAL DLL FUNCTION ARGUMENTS AND BEHAVIOR .....	13
4.4. ADDING CONTEXT HELP.....	13
<b>5. XML EXAMPLES.....</b>	<b>14</b>
5.1. SIMPLE CONTROLS .....	14
5.2. ADVANCED EDIT CONTROLS.....	14
5.3. SIMPLE BUTTON CONTROL .....	15
5.4. IMPLEMENTING STOP BUTTON.....	15
5.5. IMPLEMENTING OUTPUT WINDOW .....	16
<b>6. EXTERNAL FUNCTION EXAMPLES.....</b>	<b>17</b>
6.1. SIMPLE FUNCTION .....	17
6.2. FUNCTION PROPERLY IMPLEMENTED.....	17
6.3. BAD FUNCTION .....	18
6.4. STORING VALUES IN GLOBAL DATA.....	18

7. CONTEXT HELP EXAMPLES .....	19
7.1. CONTEXT HELP FOR A BUTTON.....	19

## 1. Overview

*XMLGUI* application is a general-purpose framework for creating customizable applications with dynamically created GUI elements defined in *xml* files. Each *xml* file contains a definition of one window that is created on startup of *XMLGUI* application.

## 2. Necessary files

Here is the list of files that must (unless noted) exist on the target system in order for *XMLGUI* to work.

Filename	Description	Location	Notes
<b>XMLGUI.exe</b>	Executable of main program.	<i>XMLGUI</i> dir	
<b>XMLParser.dll</b>	Helper dll used for accessing xml files.	<i>XMLGUI</i> dir	
<b>window.dtd</b>	Document type definition file necessary for validating xml files.	<i>XMLGUI</i> dir	Do not alter, provide as is.
<b>window_*.xml</b>	Definitions of windows are defined in these files.	<i>XMLGUI</i> dir	At least one is necessary.
<b>external dll</b>	A dll that exposes functions that actually do the work.	Usually <i>XMLGUI</i> dir unless full path is specified in XML file.	Optional but otherwise windows will have no functionality. Dynamically loaded on running functions (not on startup).
<b>msxml.dll</b>	MS XML Parser used to parse xml files.	%WINDIR%\system32 on NT family or %WINDIR%\system on 9x family	Comes with operating system or Internet Explorer 4.0+.
<b>mfc42.dll</b>	MS MFC file.	%WINDIR%\system32 on NT family or %WINDIR%\system on 9x family	Comes with operating system.

## 3. Definition of window XML files

### 3.1. Filename

Each window should be created in separate xml file, which name begins with *window\_* and ends with *.xml* extension. See examples below.

Filename	Notes
window_erase.xml	OK.
window_1.xml	OK.
windowtarget.xml	BAD – underline character missing.
window_output.txt	BAD – extension should be .xml.
my_window_new.xml	BAD – must begin with <i>window_</i> .

### 3.2. General schema

Each *xml* file can contain only one window definition. The definition of *xml* file should look generally as in the following schema:

```

<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">
<window id="XXX" attribute1="Value" attribute2="Value" ...>
  Window Name
  <control id="YYY" attribute1="Value" attribute2="Value">Caption</control>
  <item id="AAA">Value</item>
  ...
  <control id="ZZZ" attribute1="Value" attribute2="Value">Caption</control>
  ...
</window>

```

Basically there is an xml header, one `<window>` object definition that contains one or more `<control>` objects, which can contain zero or more `<item>` objects. Each object has its own attributes as defined in next sections. All attributes are case sensitive.

### 3.3. `<window>` object

This object can have value specified which will be used for widow title. The following sections specify all attributes that `<window>` object can have.

#### 3.3.1. id attribute

*Name*

**id**

*Description*

Unique window identifier. Must be unique among all windows.

*Prerequisites*

.

*Required*

Yes.

*Values*

Text value not containing spaces.

*Default value*

None – must be specified.

#### 3.3.2. initialview attribute

*Name*

### **initialview**

*Description*

Specifies if window will be visible on startup.

*Prerequisites*

.

*Required*

No.

*Values*

0 – not visible on startup

1 – visible on startup

*Default value*

1

### **3.3.3. sizeable attribute**

*Name*

**sizeable**

*Description*

Specifies if user can resize window.

*Prerequisites*

.

*Required*

No.

*Values*

0 – cannot be resized

1 – can be resized

*Default value*

1

### **3.4. <control> object**

This object can have value specified which will be used for control caption. The following sections specify all attributes that <control> object can have.

#### **3.4.1. id attribute**

*Name*

**id**

*Description*

Unique control identifier. Must be unique among all controls within a given window.

*Prerequisites*

None.

*Required*

Yes.

*Values*

Text value not containing spaces.

*Default value*

None – must be specified.

### 3.4.2. type attribute

*Name*

**type**

*Description*

Specifies the control type.

*Prerequisites*

None.

*Required*

Yes.

*Values*

- *Edit* – creates CEdit control.
- *CheckBox* – creates CButton control with BS\_AUTOCHECKBOX style.
- *Button* – creates CButton control with BS\_PUSHBUTTON style.
- *ComboBox* – creates CComboBox control.
- *SpinEdit* – creates CEdit control with attached CSpinButtonCtrl.

*Default value*

None – must be specified.

### 3.4.3. choosefile attribute

*Name*

**choosefile**

*Description*

Creates a button next to *Edit* control that will allow user to choose a file. Chosen file will be entered into *Edit* control.

*Prerequisites*

Valid only for *Edit* controls. Must not be used if **number** attribute equals to *1*.

*Required*

No.

*Values*

- 0* – no additional button will be created.
- 1* – creates additional button next to *Edit* control.

*Default value*

*0*

### 3.4.4. filemask attribute

*Name*

**filemask**

*Description*

A series of pairs that specify filters that can be applied in File Open dialog.

*Prerequisites*

Valid only for *Edit* controls and when **choosefile** attribute equals to *1*.

*Required*

No.

*Values*

- Text value having the format:  
*My files (\*.ext)|\*.ext|All files (\*.\*)|\*.\**  
The format is basically description, pipe, extension, pipe, next description etc. It is recommended to include all files mask as the last one so user can choose any file.
- If empty then All files mask will be created:  
*All files (\*.\*)|\*.\**

*Default value*

Empty.

### 3.4.5. number attribute

*Name*

**number**

*Description*

Specifies that *Edit* control can store only numeric values.

*Prerequisites*

Valid only for *Edit* controls. Must not be used if ***choosefile*** attribute equals to *1*.

*Required*

No.

*Values*

- 0* – Edit control can contain text and numeric values.
- 1* – Edit control can contain only numeric values.

*Default value*

*0*

### 3.4.6. dll attribute

*Name*

**dll**

*Description*

Defines the name of external *dll* library where user defined functions are located. This could be fully qualified path or just a filename – in this case rules apply as described in *LoadLibrary* Windows API function.

*Prerequisites*

Valid only for *Button* controls.

*Required*

No.

*Values*

Text value no longer than *MAX\_PATH*.

*Default value*

Empty.

### 3.4.7. function attribute

*Name*

**function**

*Description*

Defines the name of the function inside specified *dll* library that will be called when user hits the button.

*Prerequisites*

Valid only for *Button* controls that have *dll* attribute specified.

*Required*

No.

*Values*

Text value, case sensitive.

*Default value*

Empty.

**3.4.8. stop attribute**

*Name*

**stop**

*Description*

Identifier of another *Button* control that will be used to ask the running function to stop.

*Prerequisites*

Valid only for *Button* controls that have *dll* and *function* attributes specified.

*Required*

No.

*Values*

Text value pointing to unique control identifier of another control.

*Default value*

Empty.

**3.4.9. start attribute**

*Name*

**start**

*Description*

Identifier of another *Button* control that will be used to run an external function.

*Prerequisites*

Valid only for *Button* controls that have not *dll* attribute specified. Target *Button* control must have *dll* and *function* attributes specified.

*Required*

No.

*Values*

Text value pointing to unique control identifier of another control.

*Default value*

Empty.

**3.4.10. show attribute**

*Name*

**show**

*Description*

Identifier of a window object that will be shown when user presses the *Button* control.

*Prerequisites*

Valid only for *Button* controls.

*Required*

No.

*Values*

Text value pointing to unique window identifier.

*Default value*

Empty.

**3.4.11. readonly attribute**

*Name*

**readonly**

*Description*

Specifies that a control is read only or can be altered by user.

*Prerequisites*

None.

*Required*

No.

*Values*

0 – Control can be altered by user.

1 – Control is read only and cannot be altered by user. *Edit* controls have *ES\_READONLY* style so they are enabled, but user cannot insert there anything, and other controls have *WS\_DISABLED* style so they are effectively disabled.

*Default value*

0

**3.4.12. output attribute**

*Name*

**output**

*Description*

Specifies that given *Edit* control is the output window for receiving messages from external *dll* function. When this attribute is specified, the *Edit* control is created as maximized inside its window.

*Prerequisites*

Valid only for *Edit* controls. A *Button* control that is used for running external *dll* function must have **log** attribute specified that points to this *Edit* control. Only one control in a window can have this attribute set to 1.

*Required*

No.

*Values*

0 – Edit control does not act as a message logger

1 – Edit control acts as a message logger

*Default value*  
0

### 3.4.13. log attribute

*Name*  
**log**

*Description*  
Identifier of a window which contains an *Edit* control that will be used as output for receiving messages from external *dll* function.

*Prerequisites*  
Valid only for *Button* controls that have *dll* and *function* attributes specified. Target window must contain *Edit* control that must have *output* attribute set to 1.

*Required*  
No.

*Values*  
Text value pointing to unique window identifier.

*Default value*  
Empty.

### 3.4.14. helpid attribute

*Name*  
**helpid**

*Description*  
Specifies the numeric value of help topic.

*Prerequisites*  
None.

*Required*  
No.

*Values*  
Numeric value in decimal or hexadecimal format (0x00000).

*Default value*  
Empty.

### 3.4.15. clear attribute

*Name*  
**clear**

*Description*  
Identifier of an *Edit* control. A *Button* control that has this attribute set will clear the output *Edit* control when user hits the button. Target *Edit* control do not need to have *output* attribute set to 1, so any *Edit* control can be cleared.

*Prerequisites*  
Valid only for *Button* controls.

*Required*  
No.

*Values*

Text value pointing to unique control identifier.

*Default value*

Empty.

### 3.4.16. **timeout attribute**

*Name*

**timeout**

*Description*

Specifies the time in milliseconds that a program waits before it will kill the thread that run external *dll* function.

*Prerequisites*

Valid only when *type* attribute equals to *Button*, *dll* and *function* attributes are specified.

*Required*

No.

*Values*

- Numeric value in decimal or hexadecimal format (0x0000).
- If empty then 1 second is assumed.

*Default value*

Empty.

## 3.5. **<item> object**

This object must have value specified. Only *Edit*, *SpinEdit* and *ComboBox* controls can have these objects specified. The following sections specify all attributes that **<item>** object can have.

### 3.5.1. **id attribute**

*Name*

**id**

*Description*

Unique item identifier. Must be unique among all items within a given control.

*Prerequisites*

Valid only for *Edit* controls that have *numeric* attribute equal to 1, *SpinEdit* controls and *ComboBox* controls.

*Required*

Yes.

*Values*

- Text value not containing spaces – for *ComboBox* control specifies id of combo element.
- *Min* – for *Edit* and *SpinEdit* controls specifies the lower bound of numeric range.
- *Max* – for *Edit* and *SpinEdit* controls specifies the upper bound of numeric range.

*Default value*

None – must be specified.

## 4. Functionality

### 4.1. Startup sequence

First *DllInit* function from *XMLParser.dll* is called.

*DllInit* function searches for “*window\_\*.xml*” files in *XMLGUI* directory and for each file found it creates *CXMLWindow* object containing *CXMLControl* objects which can contain *CXMLItem* objects.

For each *CXMLWindow* object (*XMLParser.dll* side), a new *CXMLGUIDoc* object (*XMLGUI* side) is created which calls *DllGetControl* function and creates appropriate control type (*CSDControl* descendants) depending on its attributes.

### 4.2. Calling an external function

User presses a button that has a functionality of calling external *dll* function.

All controls’ identifiers and values are stored in a list of pointers (see next chapter for details).

If *log* attribute is specified program searches for output *Edit* control in given window.

Specified *dll* library is loaded into memory and a *dll* function is searched.

If the function is successfully found, a separate thread is created in which it is called.

When the thread is finished, all allocated memory is freed.

### 4.3. External dll function arguments and behavior

External *dll* function have this type and arguments definition:

```
bool DLLFUNCTION(CPtrList *pList, bool *pTerminate, CEdit *pEdit);
```

- *pList* is a pointer to list of the following objects:

```
struct CDllControl
{
    const CString sWindowId;      // window identifier
    const CString sControlId;    // control identifier
    const CString sControlValue; // control value
};
```

- *pTerminate* is a pointer to the *Boolean* value that must be checked periodically by the external function to terminate when it is set to *true*.
- *pEdit* is a pointer to output *Edit* window used as a message logger or *NULL* if it was not specified in *xml* file. External *dll* function has direct access and full control over this control.

**IMPORTANT NOTE:** To avoid memory violations, when using debug version of your external *dll* with release version of *XMLGUI* application, if you keep global copies of *CString* objects passed in *pList* argument, make sure you have actually copied them using (*LPCTSTR*) cast, and not just added reference to original *CString* object by simply assigning them to your global variables. See chapter 6.4 for further discussion of this topic.

### 4.4. Adding context help

As controls are not created by Visual Studio but during the runtime, additional work is necessary to provide context help for the windows:

- Add *helpid* attribute to `<control>` object with value greater or equal to `0x10000`.
- Edit *AfxCore.rtf* file and create topic on new page setting footnote to `HIDW_<id>`. Replace `<id>` with something unique among all controls, this could be e.g. window *id* plus control *id*.
- Edit *XMLGUI.hm* file and add link between the value created in first step and unique identifier created in step two.

## 5. XML Examples

### 5.1. Simple controls

The following example implements a sizeable window that is initially visible, has unique identifier *Test\_window* and its title is *Test panel*. It also has three controls: *Edit* control, *ComboBox* control and *CheckBox* control.

*Edit* control is enabled and user can insert there any value. Its caption is *Algorithm file:*. No additional functionality is enabled.

*ComboBox* control has three items, *Processor* identifier and *Processor type:* caption.

*CheckBox* control has *Erase* identifier, *Erase?* caption, but it is disabled (read-only).

```
<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">
<window id="Test_window" initialview="1" sizeable="1">
  Test panel
  <control id="AlgorithmFile" type="Edit">Algorithm file:</control>
  <control id="Processor" type="ComboBox">
    Processor type:
    <item id="ARM7">ARM7 Processor</item>
    <item id="ARM9">ARM9 Processor</item>
    <item id="C27x">C27x Processor</item>
  </control>
  <control id="Erase" type="CheckBox" readonly="1">Erase?</control>
</window>
```

The window will look like this on load:



### 5.2. Advanced Edit controls

Here we add an *Edit* control that has ability to choose files using specified mask.

Another *Edit* control can accept only numbers from range *1* to *32000*.

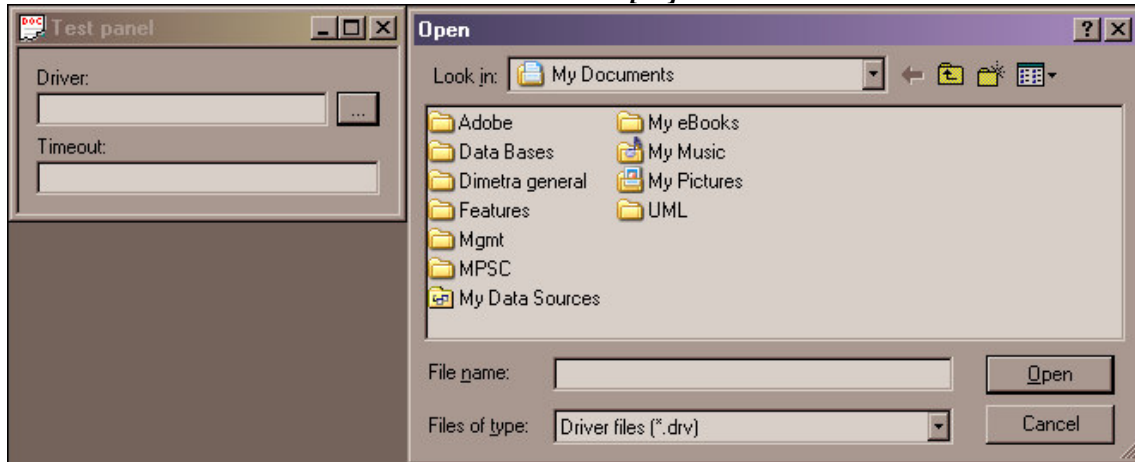
```
<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">
```

```

<window id="Test_window2" initialview="1" sizeable="1">
  Test panel
  <control id="Driver" type="Edit" choosefile="1" filemask="Driver files
    (*.drv)|*.drv|All files (*.*)|*.*" readonly="0">Driver:</control>
  <control id="Timeout" type="Edit" number="1" readonly="0">
    Timeout:
    <item id="Min">1</item>
    <item id="Max">32000</item>
  </control>
</window>

```

The window will look like this when user clicks on **[...]** button:



### 5.3. Simple Button control

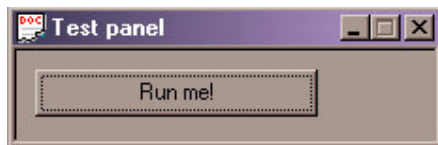
This not sizeable window will have just one button used to start *RunMe* function in *TestDll.dll* library. Note that user cannot stop this function not sees any output from it.

```

<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">
<window id="Test_window3" initialview="1" sizeable="0">
  Test panel
  <control id="Start" type="Button" dll="TestDll.dll"
    function="RunMe">Run me!</control>
</window>

```

The window will look like this on load:



### 5.4. Implementing stop button

Now lets add a stop button that will be used to stop the running external *dll* function. Note that *Start* button has pointer to *Stop* button and *vice-versa*.

```

<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">

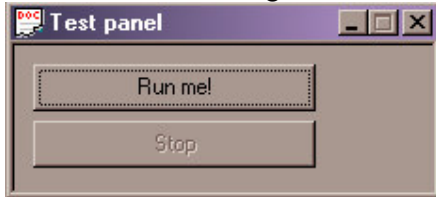
```

```

<window id="Test_window3" initialview="1" sizeable="0">
  Test panel
  <control id="Start" type="Button" dll="TestDll.dll" function="RunMe"
    stop="Stop">Run me!</control>
  <control id="Stop" type="Button" start="Start">Stop</control>
</window>

```

The window will look like this on load. Note that *Stop* button is disabled when external function is not running.



### 5.5. Implementing output window

This window will contain *Edit* control that will act as message logger for external *dll* function. It also includes a clear *Button* control so user can clear output window. Note that if a window contain output *Edit* control it must be defined as the last in order as it is resized together with client window.

```

<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">
<window id="Output_window" initialview="1" sizeable="1">
  Output window
  <control id="Clear" type="Button" clear="Output">Clear output</control>
  <control id="Output" type="Edit" output="1" />
</window>

```

We must also put a pointer to the output window to the *Start* button. Also we specify that after user clicks on *Stop* button, the program will only wait 1 second before brute killing the thread that was used to run external *dll* function.

```

<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">
<window id="Test_window3" initialview="1" sizeable="0">
  Test panel
  <control id="Start" type="Button" dll="TestDll.dll" function="RunMe"
    stop="Stop" log="Output_window" timeout="10000" >Run
    me!</control>
  <control id="Stop" type="Button" start="Start">Stop</control>
</window>

```

Now the two windows look like this.



## 6. External function examples

### 6.1. Simple function

This simple function will collect all data from the pointers list and display them in a message box. Please note that it does not use message logger feature nor checks for termination request.

```

struct CDllControl
{
    const CString sWindowId;      // window identifier
    const CString sControlId;     // control identifier
    const CString sControlValue;  // control value
};
_declspec(dllexport) void test(CPtrList *pList, bool *pTerminate, CEdit *pEdit)
{
    CString sText;
    POSITION p = pList->GetHeadPosition();
    while (p)
    {
        CDllControl *pDllControl = (CDllControl*)pList->GetNext(p);
        if (pDllControl != NULL)
        {
            sText += pDllControl->sWindowId + _T(", ") +
                pDllControl->sControlId + _T(", ") +
                pDllControl->sControlValue + _T("\n");
        }
    }

    AfxMessageBox(sText);
}

```

### 6.2. Function properly implemented

This function waits maximum 1 minute before it terminates. During that time it checks *\*pTerminate* value to know that user has requested it to finish. It also logs a message to the given output *Edit* control if it is not *NULL*. The *LogText* function does the actual logging.

```

void LogText(CEdit *pEdit, CString sText)

```

```

{
    if (pEdit == NULL)
    {
        TRACE("WARNING: pEdit is NULL!\n");
        return;
    }
    sText += _T("\r\n");
    int length = pEdit->GetWindowTextLength();
    pEdit->SetSel(length, length);
    pEdit->ReplaceSel(sText);
    pEdit->LineScroll(pEdit->GetLineCount());
}

_declspec(dllexport) void test2(CPtrList *pList, bool *pTerminate, CEdit *pEdit)
{
    int x = 0;
    CTime t = CTime::GetCurrentTime();
    CString sText;
    sText.Format("INFO: Current:%d, past:%d, terminate:%d",
    CTime::GetCurrentTime().GetMinute(), t.GetMinute(), *pTerminate);
    LogText(pEdit, sText);
    TRACE(sText + _T("\n"));
    while (CTime::GetCurrentTime().GetMinute() == t.GetMinute() && !*pTerminate)
        x++;
    if (*pTerminate)
    {
        sText = "INFO: We were asked to terminate.";
        LogText(pEdit, sText);
        TRACE(sText + _T("\n"));
    }
}
}

```

### 6.3. Bad function

This function works as in previous example but does not check termination request. This is very bad behavior which must be avoided when writing external *dll* functions. Such functions could be brute killed if not terminated on time which can lead to memory leaks.

```

_declspec(dllexport) void test3(CPtrList *pList, bool *pTerminate, CEdit *pEdit)
{
    TRACE("1NFO: Test3 start.\n");
    int x = 0;
    CTime t = CTime::GetCurrentTime();
    CString sText;
    sText.Format("2NFO: Current:%d, past:%d, terminate:%d\n",
    CTime::GetCurrentTime().GetMinute(),
    t.GetMinute(), *pTerminate);
    LogText(pEdit, sText);
    while (CTime::GetCurrentTime().GetMinute() == t.GetMinute())
        x++;
    LogText(pEdit, "3NFO: Exiting from the dll function");
    TRACE("4NFO: Test3 stop.\n");
}
}

```

### 6.4. Storing values in global data

As mentioned in chapter 4.3, global variables must keep real copies of *CString* objects passed in as *pList* argument to external function. The following example shows the bad implementation which could cause memory exceptions when debug version of external *dll* is mixed with release version of *XMLGUI* application, and the proper one which forces the *CString* object to be really copied into new one.

```

struct CDllControl
{
    const CString sWindowId;    // window identifier
    const CString sControlId;   // control identifier
    const CString sControlValue; // control value
};

CString sOurValue; // Global variable

__declspec(dllexport) void test(CPtrList * pList, bool *pTerminate, CEdit *pEdit)
{
    CString sText; // Local variable
    POSITION p = pList->GetHeadPosition();
    while (p)
    {
        CDllControl * pDllControl = (CDllControl*)pList->GetNext(p);
        if (pDllControl != NULL)
        {
            if (pDllControl->sWindowId == "WindowOne")
            {
                if (pDllControl->sControlId == "CheckBoxTwo")
                {
                    // Try to store the value of CheckBoxTwo control from WindowOne window
                    // into another variable:

                    // This is invalid as it only copies the reference to sControlValue
                    // into sOurValue string, not doing any memory copying.
                    // sOurValue = pDllControl->sControlValue;

                    // This is the proper way of copying the CString object into global
                    // variable. In this case memory will be physically copied.
                    sOurValue = (LPCTSTR)pDllControl->sControlValue;

                    // This is ok, as sText is just a local variable.
                    sText = pDllControl->sControlValue;

                    // This is also forbidden as the members of CDllControl are const.
                    // pDllControl->sControlId = "invalid";
                }
            }
        }
    }
    TRACE("sText:%s, sOurValue:%s", sText, sOurValue);
}

```

## 7. Context help examples

### 7.1. Context help for a button

First a *helpid* attribute must be added with value greater than *0x10000*. Here it is *0x10123*.

```

<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "window.dtd">
<window id="Test_window3" initialview="1" sizeable="0">
    Test panel3
    <control id="Start" type="Button" dll="TestDll.dll" function="RunMe"
        stop="Stop" log="Output_window" timeout="10000"
        helpid="0x10123" readonly="0">Run me!</control>
</window>

```

Then, a new topic must be added to *AfxCore.rtf*. The unique identifier is *HIDW\_STOP1* here.

```
if you wish to add help specific to each message  
#HIDW_STOP1 [ALIAS] section of your .HPJ file, and :  
HIDP_INVALID_FILENAME is the help  
-----  
# This is help for Start button  
  
This help is provided for Start button. It has  
two lines and some other stuff like formatting etc.  
—
```

And finally the link between the two above is added into *XMLGUI.hm* file.

```
// Frame Controls (IDW_*)  
HIDW_STOP1 0x10123
```

The help will look like this when user presses *F1* button while on a control or selects context help button from toolbar and then right clicks on a control.

