
Graphical User Interface System

for PowerTV Operating System



Document Version 1.1

February, 2002

OVERVIEW OF THE SYSTEM

This data sheet describes the Graphical User Interface System for PowerTV developed by MediaTechnik Software Group.

FEATURES

- Easy to Use Object Oriented API.
- Flexible class hierarchy, easily expandable with custom controls.
- Easily portable to all platforms supporting linear framebuffers or clippable drawing functions. (Currently supports PowerTV and Win32 system)
- XML script support for both reading and writing.
- Graphical resource editor (running on Windows).

CONTENTS

1. PLATFORM DEPENDENCY ...	2
2. XML SCRIPTS.....	2
CGENXMLSCRIPT	2
3. HELPER CLASSES.....	2
CGENCLASS	2
CGENOBJECT	3
CGUIRECT	4
4. EXAMPLES AND TUTORIALS	4
MECHANISM OF XML FILES.....	4
TUTORIAL PROGRAM.....	6
5. RESOURCE EDITOR	9

Click here to open the [Class Description Reference Manual](#).

1. PLATFORM DEPENDENCY

To compile the GUI, a C++ compiler is needed with support for inheritance, virtual member functions and static data members.

The platform dependent part of the GUI system is a very thin layer, consisting only of a few drawing operations. Every control's OnDraw() method is responsible for the look & feel of the GUI. These platform dependent function implementations are the only functions that may be used by the OnDraw() methods for drawing.

Currently there are two implementations, one for PowerTV (CGuiPowerTVDesktop) and one for Microsoft Windows (CGuiGDIDesktop).

The root control must be always an object of one of those two classes. Since all events are passed towards the root control in the hierarchy applications should derive a class from one of these. In the derived class the application can implement the OnEvent() function and process all events it is interested in.

2. XML SCRIPTS

CGenXMLScript

The GUI can be configured via XML scripts. An XML file describes the states of the GUI elements, their positions and identifier strings. Each component class has an identifier string. The XML parser creates the instances of these classes recognizing them by their identifiers. All attributes are passed to the instances by the function **SetAttrib()**.

The class CGenXMLScript parses the XML file, creates the instances of the classes and sets the attributes, making a hierarchically built object-tree of configured objects (e.g. a dialog box with lots of controls).

CGenXMLScript(const char *sName);

Class constructor, loads an XML file.

sName - name of the file with extension

virtual class CGenObject *Process(class CGenObject *pParent);

This function parses an XML file, returns the root of the built object-tree.

pParent - the parent object, that the loaded root will be linked to

3. HELPER CLASSES

CGenClass

An instance of this class represents a class derived from CGenObject. All the important information about the classes are stored in CGenClass objects. It contains generic information about a class.

m_sName

Name of the class (e.g. "CGuilmage").

m_sNick

Nickname of the class. It identifies the class in the XML file (e.g. "image" for the class CGuilmage).

m_nSize

Size of the class in bytes.

m_fCreator

Address of the function that creates an instance of the class.

m_fBaseClassProvider

Address of the function that returns a pointer to the parent class' CGenClass object.

FromName(const char *)

This static function searches for the CGenClass object with the specified name.

IsDerivedFrom(const CGenClass *pClass)

This function returns true if this class is derived from the specified class pClass.

CGenObject

This is the base class for all GUI component classes. This class provides runtime type information and serialization support for reading and writing XML files.

const CGenClass * GetThisClass

This static function returns the address of the CGenClass object associated with this class.

const CGenClass *GetRuntimeClass() const;

This function returns the runtime type descriptor CGenClass object.

IsKindOf(const CGenClass *pClass)

Returns true if this object's class is derived from pClass (casting is safe).

pClass – Pointer to the class to compare the type with.

SetAttrib(const char * sName, const char * sValue)

This virtual function is called for each attribute in an XML file. The object should process the attributes and return true if the attribute name and value are correct.

sName - the name of the property that will be set

sValue - the new value of the given property

SetParent(CGenObject * pParent)

This function is called after all attributes have been set and tells the object about its parent. It's the object's responsibility to link itself to the parent object.

pParent - pointer to the object that will be the parent

GetChild(int nIndex)

In this virtual function the object should enumerate all of its child objects. Used when storing the hierarchy in an XML file.

nIndex - index of the child

DumpAttribs(CGenXMLScriptCreator * pCreator)

This function is called when the system wants to know the attributes of the class. The object must call the DumpObject function of the specified object for each of its

attributes.

pCreator - an instance of ...

CGuiRect

This class represents a rectangular area of the GUI screen.

CGuiCtrl::CEvent [extends CGenObject]

Every event has a class derived from CGuiEvent. This class will be passed as the only parameter to OnEvent. The runtime type of this parameter will identify the event and the members of this class will provide more information about the event if necessary. A typical OnEvent function looks like this:

For example, if the client wants to handle an event of a control, it has to do the following:

```
void CMyControl::OnEvent( CEvent *pEvent )
{
    if ( pEvent->IsKindOf( CGuiEdit::CEnter::GetThisClass() ) )
    {
        // process editbox event (enter pressed)
        ...
    }
    if ( pEvent->IsKindOf( .... ) )
    {
    };
};
```

4. EXAMPLES AND TUTORIALS**Mechanism of XML files**

This tutorial will create a little GUI application that waits for a login name and a password, which the user must enter. After that a button must be pressed to close the window.

The XML file that describes this view looks like this:

```
<image pos="170 140 50 50" bitmap="window_connect.IMG"
id="signonwindow">
  <view pos="11 66 245 94" background="false">
    <edit pos="0 0 0 20" id="acc_email"/>
  </view>
  <view pos="11 126 245 154" background="false">
    <edit pos="0 0 0 20" id="acc_passwd"/>
  </view>
</image>
```

Attributes:

image

The XML parser will create an instance of the CGuiImage class.

pos="170 140 50 50"

The rectangle that will be tile-mapped with the image.

bitmap="window_connect.IMG"

The image file name.

id="signonwindow"

A string that identifies the control. Can be used to get a pointer to the control via the GetByID() function.

view

The XML parser will create an instance of the CGuiView class.

pos="11 66 245 94"

The rectangle that will be the area of the view.

background="false"

Controls if background should be drawn in the area of the control or not.

edit

The XML parser will create an instance of the CGuiEdit class.

id="acc_email"

The ID of the editline will be acc_email.

To parse this XML script, the client program has to do the following steps:

```
CGenXMLScript( "signon.xml" ).Process( m_pDesktop );
```

This line of code loads the file "signon.xml" and processes it. The pointer m_pDesktop tells the control that it will be the parent of the new control that is defined by the XML file. The user can type text into the edit lines.

If the client wants to access the texts that were entered into the edit boxes:

```
((CGuiEdit *)GetByID( "acc_email" ))->m_sText;
((CGuiEdit *)GetByID( "acc_passwd" ))->m_sText;
```

Tutorial program

This example program will create a more complex GUI application.

This is the program's working mechanism, step by step:

```
class CSampleApp :
#ifdef _MSC_VER
    public CGuiGDIDesktop
{
public:
    CSampleApp();
#else
    public CGuiPowerTVDesktop
{
public:
    CSampleApp( Pk_Queue* );
    CTimer* m_pTimer;
#endif
```

The application has declared a class that will be the application's class. It has to extend the desktop class of the given platform. This example program will be compiled for the Microsoft Windows platform if VisualC++ is used and for the PowerTV platform otherwise.

```
void OnEvent( CGuiCtrl::CEvent * );
```

```
bool OnKey( int, bool );
```

We want to catch GUI events, so we override the OnEvent() and the OnKey() functions.

```
int m_nBigFont;
```

This variable will store the identifier of the font set.

```
class CGuiTextBuffer *m_pLog;
```

The sample program displays the events in a text box at the bottom of the screen. This pointer is used to access this text box control.

```
#ifdef _MSC_VER
CSampleApp::CSampleApp( void ) : CGuiGDIDesktop( AfxGetApp()-
>m_pMainWnd, 640, 480 )
```

Constructor header for win32 version.

```
#else
CSampleApp::CSampleApp( Pk_Queue* pAppQueue ) :
CGuiPowerTVDesktop( win_GetGrafPort(0, kWin_System), 640, 480
)
#endif
```

Constructor header for PowerTV version.

Redraw();

To avoid the 'framebuffert trash' on the screen, we refresh the entire display.

```
#ifndef _MSC_VER
// class init
CGuiWindow::cClass.Init();
CGuiButton::cClass.Init();
CGuiView::cClass.Init();
CGuiEdit::cClass.Init();
CGuiText::cClass.Init();
CGuiList::cClass.Init();
CGuiTextBuffer::cClass.Init();
CGuiTab::cClass.Init();
CGuiSheet::cClass.Init();
CGuiImage::cClass.Init();
CGuiCtrl::cClass.Init();
#endif
```

These init function calls should be only in the PowerTV version. On Win32 this is done by the constructors of these classes' global instances (not called on PowerTV).

```
// control init
CGuiButton::Init();
CGuiView::Init();
CGuiTab::Init();

CGenXMLScript cScript0( "base.xml" );
cScript0.Process( this );
```

This two line of code loads the XML file "base.xml", and parses it. The Process() call's parameter (this) will be the parent of the GUI controls, so our application class will be the control object-hierarchy root.

SetFocus();

```
m_pLog = (CGuiTextBuffer *)GetByID( "log" );
```

This line of code access the log window from the hierarchy, and store it as a member function of the application class. The identifier of the object is used to access its address.

```
CGuiList *pL = (CGuiList *)GetByID( "list" );
pL->AddItem( "first" );
pL->AddItem( "second" );
pL->AddItem( "third" );
pL->AddItem( "fourth" );
pL->AddItem( "fifth" );
pL->AddItem( "sixth" );
```

```
pL->AddItem( "seventh" );
```

The first line gets the address of the object with the identifier 'list'. This is a listbox control, the other lines add items into it.

CheckForRedraw();

Redraw the required areas.

```
#ifndef _MSC_VER
    printf("\nCreating timer...");
    m_pTimer = new CTimer(pAppQueue, CSampleCore::Tick);
    m_pTimer->RegisterInterest();
    m_pTimer->Activate();
    printf("Timer activated.");
#endif
```

Needs on PowerTV, for using timer functions.

void CSampleApp::OnEvent(CGuiCtrl::CEvent *pEvent)

We override this virtual function to handle GUI events.

```
    if ( pEvent->IsKindOf(
CGuiButton::CPushed::GetThisClass() ))
    {
        m_pLog->AddText( "Sample button pushed." );
    };
```

This code recognizes if the button is pushed.

```
    if ( pEvent->IsKindOf( CGuiEdit::CEnter::GetThisClass() )
)
    {
        char sT[0x80];
        sprintf( sT, "Enter pressed in editbox (%s)",
((CGuiEdit *)pEvent->m_pRaiser)->m_sText );
        m_pLog->AddText( sT );
    };
```

This code recognizes if the enter was pressed in the editbox.

```
    if ( pEvent->IsKindOf(
CGuiList::CSelected::GetThisClass() ))
    {
        char sT[0x80];
        sprintf( sT, "List %d. row selected", ((CGuiList
*)pEvent->m_pRaiser)->m_nSelectedLine );
        m_pLog->AddText( sT );
    };
};
```

This code recognizes if an item is selected from the listbox.

