

GUI Interface for *Project54* Applications Using XML Defined GUI Screens W. Thomas Miller, III

This document describes the software interface for applications supporting *Project54* GUI interfaces defined in XML files. A separate document describes the specific XML syntax and underlying functionality for the GUI interfaces.

In order to use the software procedures described in this document, an application project should replace the normal *P54AppBase.cpp* file with the *P54XMLAppBase.cpp* file. In addition, an application project should include the *P54XMLAppBase.h* file which defines the procedure prototypes listed below. Finally, an application project no longer needs to include the *P54Guilib.h* and *P54Guilib.lib* library files, unless it is necessary for the application to explicitly call procedures contained in the traditional *Project54* GUI API.

Standard Routines Called by the Application Specific Code:

Generally, a custom *Project54* application creates and interacts with its GUI using the following two procedures, defined in *P54XMLAppBase.h* and implemented in *P54XMLAppBase.cpp*.

```
HRESULT CreateXMLGui(LPWSTR gui_file_path,  
                    LPWSTR gui_name);
```

The above procedure creates a new GUI using the XML description in the file indicated. *gui_file_path* is a pointer to a string containing the full path to the xml file describing the GUI. If successful, the procedure returns the value S_OK and copies the logical name of the GUI (the “id” attribute of the GUI <screen> XML element) to the string variable pointed to by *gui_name*. The routine returns a value other than S_OK if the GUI creation fails. An application can create and manage multiple GUIs, differentiated by their *gui_name* strings.

```
HRESULT MessageToGui(LPWSTR source,  
                    LPWSTR gui_name,  
                    LPWSTR messageID,  
                    LPWSTR mtext);
```

The above procedure sends a message to the named GUI. This message *source*, *messageID* and *mtext* strings are processed as explained in the XML GUI specification document. If successful, the procedure returns the value S_OK. The routine returns a value other than S_OK if the message delivery fails. This is not necessarily an error since it may just indicate that the GUI named had no items which respond to the specific message sent.

Standard Routine Implemented by the Application Specific Code:

Most messages generated by the application GUI (as explained in the XML GUI specification document) are placed on the application's normal message queue, and are thus fielded in the application's standard message handling procedure:

```
void AppHandleMessage(LPWSTR source,  
                     LPWSTR dest,  
                     LPWSTR messageID,  
                     LPWSTR message_text);
```

Generally, the source and destination strings in these messages from the GUI will both be equal to *gui_name* to identify which of the application's GUIs produced the message. These messages are automatically looped back to the GUI itself, so the application specific code should only field those messages which trigger application specific functionality. If the GUI generates a message addressed explicitly to a different application (via an <action> element in the GUI script) that message is delivered transparently to the Application Manager interface for delivery to the destination application, and is not seen by the parent application.

Optional Routines Called by the Application Specific Code:

Optionally, a *Project54* application can create its GUI using the following procedure.

```
HRESULT CreateXMLGui(LPWSTR gui_file_path,  
                    LPWSTR app_registry_name,  
                    LPWSTR app_messaging_name,  
                    LPWSTR gui_name);
```

The above procedure creates a new GUI using the XML description in the file indicated. *gui_file_path* is a pointer to a string containing the full path to the xml file describing the GUI. *app_registry_name* is a pointer to a string containing the name of the registry path to be used to resolve GUI script registry references (which defaults to the *AppName* string variable of the application when using the standard 2 argument form of the call). *app_messaging_name* is a pointer to a string containing the messaging name to be used as the *self* name in messages sent to other applications (which defaults to the *self* string variable of the application when using the standard 2 argument form of the call). If successful, the procedure returns the value S_OK and copies the logical name of the GUI (the "id" attribute of the GUI <screen> XML element) to the string variable pointed to by *gui_name*. The routine returns a value other than S_OK if the GUI creation fails.

The following two procedures are only needed if it is necessary to control the GUI window or control elements in a manner which is not supported by the XML script rules.

```
HRESULT GetGuiHandle(LPWSTR gui_name,  
                    HWND * hwGui);
```

This routine returns the window handle of the named GUI control element. *gui_name* is the logical name of the GUI (the “id” attribute of the GUI <screen> XML element). The handle returned to *hwGui* is both the *Project54* GUI window handle and the Windows OS window handle, and can thus be used in *Project54* GUI API calls or in Windows API calls which require a window handle. If successful, the procedure returns the value S_OK. The routine returns a value other than S_OK if the named GUI is not found.

```
HRESULT GetGuiElementHandle(LPWSTR gui_name,  
                            LPWSTR element_name,  
                            int * hElement);
```

This routine returns the integer handle of the named GUI control element in the named GUI. *gui_name* is the logical name of the GUI (the “id” attribute of the GUI <screen> XML element). *element_name* is the logical name of the GUI control element (the “id” attribute of the specific GUI XML element). The handle returned to *hElement* (along with the GUI window handle) can then be used in normal *Project54* GUI API calls to provide fine control of the GUI behavior. If successful, the procedure returns the value S_OK. The routine returns a value other than S_OK if the named control is not found.

Optional Callback Routines Implemented in the Application Specific Code:

Two callback routines can be implemented in the application specific code in order to field and handle either *Project54* GUI interaction events or Windows OS window procedure messages. In order to use one or both of these callback mechanisms, the file *P54XMLAppBase.h* must be edited to define the appropriate control constants (these definitions are commented out in the header file by default):

```
#define SUPPORT_GUI_CALLBACK  
#define SUPPORT_WINDOW_CALLBACK  
  
bool appGuiCallback(HWND hwGui,  
                   UINT hElement);
```

This callback procedure (if enabled) is called from the standard *Project54* GUI event loop before the event is processed by the XML GUI logic. *hwGui* is the handle of the *Project54* GUI window and *hElement* is the handle of the activated *Project54* GUI control. The application can use this to process GUI control events explicitly using the normal *Project54* GUI API. If the application specific callback procedure returns *true*, GUI event calls to the callback procedure will continue. If the application specific callback procedure returns *false*, GUI event calls to the callback procedure will be discontinued.

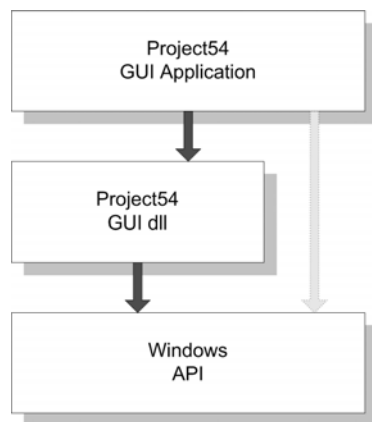
```

bool appWindowCallback(HWND hwGui,
                       UINT message,
                       WPARAM wParam,
                       LPARAM lParam);

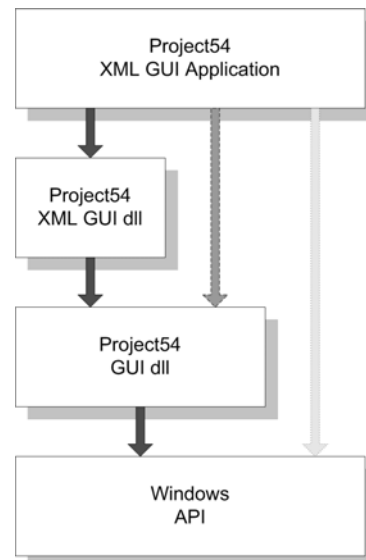
```

This callback procedure (if enabled) is called from the Windows OS standard window message processing thread, which is otherwise hidden inside the GUI dll. All four arguments are as provided by the Windows OS and as described in the Windows API. It is called before the Windows window message is processed inside the GUI dll. The application specific callback procedure must return *true* to continue receiving Windows OS window messages. The procedure can return *false* to discontinue handling Windows OS window messages.

XML GUI Run-time Environment



Project54 Application with Traditional GUI



Project54 Application with XML GUI

The figure above illustrates the run-time environments for *Project54* applications which use the traditional GUI library and those that use the XML GUI support. The XML GUI routines are implemented in a single COM component dll. These routines call on the standard *Project54* GUI API routines which are implemented in a separate COM component dll. Finally, the standard *Project54* GUI routines call on the standard Windows OS API to paint the windows and manage the GUI interactions. This organization has two important implications:

1. *Project54* applications which use the XML GUI approach are completely compatible with older *Project54* applications which use the GUI API calls directly, since both GUI implementations rely on the same underlying COM component to manage the individual application GUIs. The look and feel of all *Project54* GUIs is still controlled by a single COM component.

2. *Project54* applications which use the XML GUI approach can augment their functionality by directly calling routines in the standard *Project54* GUI API or in the Windows OS user interface API. At the extreme, an XML script could be used to layout a GUI screen with no inherent functionality, and all functionality could then be achieved via standard *Project54* GUI API calls. However, this should generally be avoided if similar GUI behavior can be achieved using the functional features of the XML GUI script.