Columns UI SDK

Reupen Shah

Feb 12, 2022

USER GUIDE

CHAPTER

ONE

GETTING STARTED

The Columns UI SDK provides interfaces you can use to:

- Create windows controlled by a host and embedded in the host's window
- Provide information about commands to be used as a toolbar button

1.1 Installation

You'll need:

- Microsoft Visual Studio 2019
- foobar2000 SDK

To install, extract the columns_ui-sdk.7z archive to the foobar2000 subdirectory of your foobar2000 SDK folder.

1.2 Usage

Insert the columns_ui-sdk project into your solution, and add it as a dependency for your project. Then `#include "columns_ui-sdk/ui_extension.h"` in your project as needed.

1.3 Examples

Two examples are included in the SDK:

- foo_uie_example – a simple panel that displays some text and implements a context menu item
- foo_uie_console – a console viewer

1.4 Panel APIs

1.4.1 APIs

Clients should implement *uie::window*. Specific sub-classes exist for

- Menus: *uie::menu_window*
- Playlists: *uie::playlist_window*

- Splitter panels: *uie::splitter_window*

Hosts should implement `uie::window_host`. Hosts wishing to expose external control methods can implement `uie::window_host_with_control` instead.

1.4.2 Helpers

The preferred method of implementing the window class is to derive from *uie::container_ui_extension* (although this may not be suitable for single-instance panels or dialog-based panels).

1.5 Button APIs

1.5.1 APIs

The base class for buttons is *uie::button*.

If you wish to provide default bitmaps and additional information for your menu items, derive from *uie::menu_button*. If you wish to implement a custom button not based upon a menu item, derive from *uie::custom_button*.

1.5.2 Standard windows

The GUIDs for the standard panels may be found in the *cui::panels* namespace. The GUIDs for the standard toolbars may be found in the *cui::toolbars* namespace.

You may use these GUIDs to create the standard windows in your own component; do not use them as GUIDs for your own windows.

CHAPTER

TWO

BASE EXTENSION

class `uie::extension_base` : public service_base
Base class for *uie::window* and *uie::visualisation* classes.

Subclassed by *uie::visualisation*, *uie::window*

Public Functions

virtual const GUID &`get_extension_guid`() const = 0
Get unique ID of extension.

This GUID is used to identify a specific extension.

Returns extension GUID

virtual void `get_name`(pfc::string_base &out) const = 0
Get a user-readable name of the extension.

See also:

get_extension_guid

> **Warning:** Do not use the name to identify extensions; use extension GUIDs instead.

Parameters `out` – **[out]** receives the name of the extension, e.g. "Spectrum analyser"

inline virtual void `set_config`(stream_reader *p_reader, t_size p_size, abort_callback &p_abort)
Set instance configuration data.

Remark

- Only called before enabling/window creation.
- Must not be used by single instance extensions.
- You should also make sure you deal with the case of an empty stream

---

Throws `Throws` – pfc::exception on failure

Parameters

- **p_reader** – **[in]** Pointer to configuration data stream
- **p_size** – **[in]** Size of data in stream
- **p_abort** – **[in]** Signals abort of operation

inline virtual void **get_config**(stream_writer *p_writer, abort_callback &p_abort) const
Get instance configuration data.

Remark

Must not be used by single instance extensions.

Note: Consider compatibility with future versions of you own component when deciding upon a data format. You may wish to change what is written by this function in the future. If you prepare for this in advance, you won't have to take measures such as changing your extension GUID to avoid incompatibility.

Throws Throws – pfc::exception on failure

Parameters

- **p_writer** – **[out]** Pointer to stream receiving configuration data
- **p_abort** – **[in]** Signals abort of operation

inline virtual void **import_config**(stream_reader *p_reader, t_size p_size, abort_callback &p_abort)
Set instance configuration data. This differs from set_config in that the data will be of that returned by export_config.

Remark

- Only called before enabling/window creation.

Note: The default implementation calls set_config for compatibility only. Be sure that you override if you need to.

Throws Throws – pfc::exception on failure

Parameters

- **p_reader** – **[in]** Pointer to configuration data stream
- **p_size** – **[in]** Size of data in stream
- **p_abort** – **[in]** Signals abort of operation

inline virtual void **export_config**(stream_writer *p_writer, abort_callback &p_abort) const
Get instance configuration data. This differs from get_config, in that what is written is intended to be transferable between different foobar2000 installations on different computers (i.e. self-contained).

Note: The default implementation calls get_config for compatibility only. Be sure that you override if you need to.

Throws Throws – pfc::exception on failure

Parameters

- **p_writer** – **[out]** Pointer to stream receiving configuration data
- **p_abort** – **[in]** Signals abort of operation

inline virtual bool **have_config_popup**() const
Gets whether the extension has a modal configuration window.

The window is exposed through *show_config_popup()*

Returns true iff a configuration window is exposed through show_config_popup

inline virtual bool **show_config_popup**(HWND wnd_parent)
Displays a modal configuartion dialog.

Parameters **wnd_parent** – **[in]** The window to use as the owner window for your configuration dialog

Returns false if the configuration did not change

inline virtual void **get_menu_items**(*menu_hook_t* &p_hook)
Retrieve menu items to be displayed in the host menu.

Parameters **p_hook** – **[in]** The interface you use to add your menu items

void **set_config_from_ptr**(const void *p_data, t_size p_size, abort_callback &p_abort)
Helper function, set instance configuration data from raw pointer.

See also:

set_config

Throws Throws – pfc::exception on failure

Parameters

- **p_data** – **[in]** Pointer to configuration data
- **p_size** – **[in]** Size of data
- **p_abort** – **[in]** Signals abort of operation

void **import_config_from_ptr**(const void *p_data, t_size p_size, abort_callback &p_abort)
Helper function. Import instance configuration data from a raw pointer.

See also:

import_config.

Throws `Throws` – pfc::exception on failure

Parameters

- **p_data** – **[in]** Pointer to configuration data
- **p_size** – **[in]** Size of data in stream
- **p_abort** – **[in]** Signals abort of operation

void **get_config_to_array**(pfc::array_t<uint8_t> &p_data, abort_callback &p_abort, bool b_reset = false) const

Helper function, writes instance configuration data to an existing array.

See also:

get_config

Throws `Throws` – pfc::exception on failure

Parameters

- **p_data** – **[out]** Array receiving configuration data
- **p_abort** – **[in]** Signals abort of operation
- **b_reset** – **[in]** Indicates whether the contents of the array should first be cleared

pfc::array_t<uint8_t> **get_config_as_array**(abort_callback &p_abort) const

Helper function, writes instance configuration data to a new array.

See also:

get_config

Throws `Throws` – pfc::exception on failure

Parameters **p_abort** – **[in]** Signals abort of operation

void **export_config_to_array**(pfc::array_t<uint8_t> &p_data, abort_callback &p_abort, bool b_reset = false) const

Helper function, exports instance configuration data to an array.

See also:

export_config

Throws `Throws` – pfc::exception on failure

Parameters

- **p_data** – **[out]** Array receiving exported configuration data
- **p_abort** – **[in]** Signals abort of operation
- **b_reset** – **[in]** Indicates whether the contents of the array should first be cleared

CHAPTER

THREE

WINDOW

These interfaces are used to implement panels and toolbars.

3.1 Window

class `uie::window` : public uie::*extension_base*
Interface for window service.

Subclassed by *uie::container_ui_extension_t< W, T >*, *uie::menu_window*, *uie::playlist_window*, *uie::splitter_window*

Public Functions

virtual const bool **`get_is_single_instance`**`()` const = 0
Gets whether the panel is single instance or not.

Note: Do not explicitly override. The service factory implements this method.

virtual void **`get_category`**`(`pfc::string_base &out`)` const = 0
Gets the category of the extension.

Categories you may use are "Toolbars", "Panels", "Splitters", "Playlist views" and "Visualisations"

Parameters **`out`** – **[out]** receives the category of the panel, utf-8 encoded

inline virtual bool **`get_short_name`**`(`pfc::string_base &out`)` const
Gets the short, presumably more user-friendly than the name returned by get_name, name of the panel.

Parameters **`out`** – **[out]** receives the short name of the extension, e.g. "Order" instead of "Playback order", or "Playlists" instead of "Playlist switcher"

Returns true if the extension has a short name

inline virtual bool **`get_description`**`(`pfc::string_base &out`)` const
Gets the description of the extension.

Parameters **`out`** – **[out]** receives the description of the extension, e.g. "Drop-down list for displaying and changing the current playback order"

Returns true if the extension has a description

virtual unsigned **get_type**() const = 0
Gets the type of the extension.

See also:

uie::window_type_t

Returns a combination of uie::type_* flags

inline virtual bool **get_prefer_multiple_instances**() const
Gets whther the panel prefers to be created in multiple instances.

For example, a spacer panel.

Returns true iff the panel prefers to be created in multiple instances

virtual bool **is_available**(const window_host_ptr &p_host) const = 0
Get availability of the extension.

This method is called before create_or_transfer() to test, if this call will be legal. If this instance is already hosted, it should check whether the given host's GUID equals its current host's GUID, and should return `false`, if it does. This is mostly important for single instance extensions.

Extensions that support multiple instances can generally return `true`.

Returns whether this instance can be created in or moved to the given host

virtual HWND **create_or_transfer_window**(HWND wnd_parent, const window_host_ptr &p_host, const ui_helpers::window_position_t &p_position = ui_helpers::window_position_null) = 0
Create or transfer extension window.

Create your window here.

In the case of single instance panels, if your window is already created, you must (in the same order):

- Hide your window. i.e:

```
ShowWindow(wnd, SW_HIDE)
```

- Set the parent window to to wnd_parent. I.e.

```
SetParent(get_wnd(), wnd_parent)
```

- Move your window to the new window position. I.e.:

```
SetWindowPos(get_wnd(), NULL, p_position.x, p_position.y, p_position.cx, p_
↪position.cy, SWP_NOZORDER);
```

- Call relinquish_ownership() on your current host.

Other rules you should follow are:

- Ensure you are using the correct window styles. The window MUST have the WS_CHILD window style. It MUST NOT have the WS_POPUP, WS_CAPTION styles.
- The window must be created hidden.
- Use WS_EX_CONTROLPARENT if you have child windows that receive keyboard input, and you want them to be included in tab operations in the host window.

- Do not directly create a common control as your window. You must create a window to contain any common controls, and any other controls that communicate to the parent window via WM_COMMAND and WM_NOTIFY window messages.
- Under NO CIRCUMSTANCES may you subclass the host window.
- If you are not hosting any panels yourself, you may dialog manage your window if you wish.
- The window MUST have a dialog item ID of 0.

Parameters

- **wnd_parent** – **[in]** Handle to the window to use as the parent for your window
- **p_host** – **[in]** Pointer to the host that creates the extension. This parameter may not be NULL.
- **p_position** – **[in]** Initial position of the window

Pre May only be called if *is_available()* returned true.

Returns Window handle of the panel window

virtual void **destroy_window**() = 0
Destroys the extension window.

virtual HWND **get_wnd**() const = 0
Gets extension window handle.

Pre May only be called on hosted extensions.

Returns Window handle of the extension window

inline virtual void **get_size_limits**(size_limit_t &p_out) const
Gets size limits of the window.

Override if you like, or just handle WM_GETMINMAXINFO.

Note: This function is reserved for future use. Handle WM_GETMINMAXINFO for now instead.

Parameters **p_out** – **[out]** Receives the size limits of the window.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*window*)

Public Static Functions

static inline bool **create_by_guid**(const GUID &guid, window_ptr &p_out)
Creates extension by GUID.

Parameters

- **guid** – **[in]** GUID of a ui_extension
- **p_out** – **[out]** Receives a pointer to the window.

Returns true if the window was found and instantiated. You may assume that if the method returns true, p_out is a valid pointer.

static HWND **g_on_tab**(HWND wnd_focus)
Helper function. Activates next or previous window.

Parameters **wnd_focus** – **[in]** Window you want the next or previous window handle respective to.

Returns The handle to the window that was activated, or NULL if none was.

static bool **g_process_keydown_keyboard_shortcuts**(WPARAM wp)
Helper function. Processes keyboard shortcuts using keyboard_shortcut_manager_v2::process_keydown_simple(). Requires foobar2000 >= 0.9.5.

Parameters **wp** – **[in]** Key down message WPARAM value.

Returns If a shortcut was executed.

3.2 Playlist view

class **uie::playlist_window** : public uie::*window*
Subclass of *uie::window* for playlist views.

Public Functions

virtual void **set_focus**() = 0
Called by host to indicate you should focus your window.

Pre May only be called on hosted extensions.

FB2K_MAKE_SERVICE_INTERFACE(*playlist_window*, *window*)

3.3 Menu

class **uie::menu_window** : public uie::*window*
Subclass of *uie::window*, specifically for menu bars.

Subclassed by *uie::menu_window_v2*

Public Functions

virtual bool **on_menuchar**(unsigned short chr) = 0
Called by host when a menu accelerator is pressed.

Called by host in its WM_MENUCHAR handler to notify extension that a menu was requested to be opened. You should check whether the accelerator key pressed is one of yours.

Parameters **chr** – **[in]** character that was pressed

Pre May only be called on hosted extensions.

Returns whether you claimed the accelerator key, and showed/will show your menu

virtual void **set_focus**() = 0
Called by host to indicate you should focus your menu.

Pre May only be called on hosted extensions.

virtual bool **is_menu_focused**() const = 0
Retrieve whether the menu has the keyboard focus..

Pre May only be called on hosted extensions.

Returns whether your menu has keyboard focus

virtual void **show_accelerators**() = 0
Indicates that you should underline menu access keys in your menu.

Remark

Applicable only if your menu underlines menu access keys only when activated by the keyboard. This is typically determined by the SPI_GETKEYBOARDCUES system parameter.

Remark

Do not change the state within this function call. Use PostMessage.

Implementation example

```
PostMessage(wnd_menu, WM_UPDATEUISTATE, MAKEWPARAM(UIS_CLEAR , UISF_
↪HIDEACCEL), 0);
```

virtual void **hide_accelerators**() = 0
Indicates that you should stop underlining menu access keys in your menu.

Remark

Applicable only if your menu underlines menu access keys only when activated by the keyboard. This is typically determined by the SPI_GETKEYBOARDCUES system parameter.

Remark

Do not change the state within this function call. Use PostMessage.

Implementation example

```
 BOOL b_showkeyboardcues = TRUE;
 SystemParametersInfo(SPI_GETKEYBOARDCUES, 0, &b_showkeyboardcues, 0);
 PostMessage(wnd_menu, WM_UPDATEUISTATE, MAKEWPARAM(b_showkeyboardcues ?␣
↪UIS_CLEAR : UIS_SET , UISF_HIDEACCEL),
0);
```

FB2K_MAKE_SERVICE_INTERFACE(*menu_window*, *window*)

class uie::**menu_window_v2** : public uie::*menu_window*
Subclass of *uie::menu_window*, with additional functions.

Public Functions

virtual HWND **get_previous_focus_window**() const = 0
Retrieve handle of the window that was focused before the menu was.

Implementations should track the previously focused window using the WM_SETFOCUS and WM_KILLFOCUS window messages.

Pre May only be called on hosted extensions.

Returns HWND of the previously focused window, or nullptr if no such window or the menu bar is not currently focused.

FB2K_MAKE_SERVICE_INTERFACE(*menu_window_v2*, *menu_window*)

3.4 Factories

template<class **T**>

class uie::**window_factory**
Service factory for multiple instance windows.

Usage example

```
static window_factory< my_uie > foo_extension;
```

Public Functions

inline **window_factory**()

inline **~window_factory**()

inline virtual void **instance_create**(service_ptr_t<service_base> &p_out)

CHAPTER

FOUR

SPLITTER WINDOW

These interfaces are used to implement panels that can host other panels.

4.1 Splitter

class `uie::splitter_window` : public uie::*window*

Subclass of *uie::window*, specifically for splitters.

Splitter classes must support multiple instances

Subclassed by *uie::splitter_window_v2*

Public Functions

inline virtual bool **get_config_item_supported**(t_size p_index, const GUID &p_type) const

Get config item supported.

Returns count

inline virtual bool **get_config_item**(t_size index, const GUID &p_type, stream_writer *p_out, abort_callback &p_abort) const

Creates non-modal child configuration dialog. Since its non-modal, remember to keep a refcounted reference to yourself. Use WS_EX_CONTROLPARENT.

inline bool **get_config_item**(t_size index, const GUID &p_type, stream_writer *p_out) const

inline virtual bool **set_config_item**(t_size index, const GUID &p_type, stream_reader *p_source, abort_callback &p_abort)

template<typename **class_t**>
inline bool **set_config_item_t**(t_size index, const GUID &p_type, const *class_t* &p_val, abort_callback &p_abort)

template<class **T**>
inline bool **get_config_item**(t_size p_index, const GUID &p_type, *T* &p_out, abort_callback &p_abort) const

template<class **T**>
inline bool **get_config_item**(t_size p_index, const GUID &p_type, *T* &p_out) const

virtual void **insert_panel**(t_size index, const *splitter_item_t* *p_item) = 0
This method may be called on both active and inactive (i.e. no window) instances

virtual void **remove_panel**(t_size index) = 0
This method may be called on both active and inactive (i.e. no window) instances

virtual void **replace_panel**(t_size index, const *splitter_item_t* *p_item) = 0
This method may be called on both active and inactive (i.e. no window) instances

virtual t_size **get_panel_count**() const = 0

inline virtual t_size **get_maximum_panel_count**() const

inline virtual void **register_callback**(class splitter_callback *p_callback)
Reserved for future use

inline virtual void **deregister_callback**(class splitter_callback *p_callback)
Reserved for future use

inline void **get_panel**(t_size index, pfc::ptrholder_t<*splitter_item_t*> &p_out) const

inline t_size **add_panel**(const *splitter_item_t* *p_item)

inline void **swap_items**(t_size p_item1, t_size p_item2)

inline bool **move_up**(t_size p_index)

inline bool **move_down**(t_size p_index)

inline bool **find_by_ptr**(const uie::*window*::ptr &window, t_size &p_index)

inline void **remove_panel**(const uie::*window*::ptr &window)

inline bool **set_config_item**(t_size index, const GUID &p_type, const void *p_data, t_size p_size,
abort_callback &p_abort)

FB2K_MAKE_SERVICE_INTERFACE(*splitter_window*, *window*)

Public Static Attributes

static const GUID **bool_show_caption** = {0x4673437d, 0x1685, 0x433f, {0xa2, 0xcc, 0x38, 0x64, 0xd6, 0x9, 0xf4, 0xe2}}

static const GUID **bool_hidden** = {0x35fa3514, 0x8120, 0x49e3, {0xa5, 0x6c, 0x3e, 0xa1, 0xc8, 0x17, 0xa, 0x2e}}

static const GUID **bool_autohide** = {0x40c95dfe, 0xe5e9, 0x4f11, {0x90, 0xec, 0xe7, 0x41, 0xbe, 0x88, 0x7d, 0xdd}}

static const GUID **bool_locked** = {0x3661a5e9, 0xfb4, 0x4d2a, {0xac, 0x5, 0xef, 0x2f, 0x47, 0xd1, 0x8a, 0xd9}}

static const GUID **uint32_orientation** = {0x709465de, 0x42cd, 0x484d, {0xbe, 0x8f, 0xe7, 0x37, 0xf0, 0x1a, 0x64, 0x58}}

static const GUID **bool_show_toggle_area** = {0x5ce8945e, 0xbbb4, 0x4308, {0x99, 0xc1, 0xdf, 0xa6, 0xd1, 0xf, 0x90, 0x4}}

static const GUID **uint32_size** = {0x5cb327ab, 0x34eb, 0x409c, {0x9b, 0x4e, 0x10, 0xd0, 0xa3, 0xb0, 0x4e, 0x8d}}

static const GUID **bool_use_custom_title** = {0x71bc1fbc, 0xedd1, 0x429c, {0xb2, 0x62, 0x74, 0xc2, 0xf0, 0xa, 0xb3, 0xd3}}

static const GUID **string_custom_title** = {0x3b4deda5, 0x493d, 0x4c5c, {0xb5, 0x2c, 0x3, 0x6d, 0xe4, 0xcf, 0x43, 0xd9}}

static const GUID **size_and_dpi** = {0x443eea36, 0xe5f0, 0x4add, {0xba, 0xe, 0xf3, 0x17, 0x26, 0xb0, 0xbc, 0x45}}

class `uie::splitter_window_v2` : public uie::*splitter_window*

Extends *uie::splitter_window*, providing additional methods used for live editing.

New in SDK version 6.5.

Public Functions

inline virtual bool **is_point_ours**(HWND wnd_point, const POINT &pt_screen,
pfc::list_base_t<uie::*window*::ptr> &p_hierarchy)

Checks if a point is within this splitter window. Used for live layout editing.

If the point is within your window (including any child windows), append yourself to p_hierarchy. If it is in a non-splitter child window, additionally append the child window to the list. If the child window is a splitter window, call its is_point_ours to complete the hierarchy.

Parameters

- **wnd_point** – **[in]** The window the original mouse message was being sent to.
- **pt_screen** – **[in]** The point being checked.

- **p_hierarchy** – **[out]** Receives the hierarchy of windows leading to the point including this window.

Returns True if the point is window the window; otherwise false.

inline virtual void **get_supported_panels**(const pfc::list_base_const_t<uie::*window*::ptr> &p_windows, bit_array_var &p_mask_unsupported)

Checks if windows can be inserted into this splitter. Used for live editing.

Implement this by calling *uie::window::is_available* on each window.

Parameters

- **p_windows** – **[in]** List of windows to check.
- **p_mask_unsupported** – **[out]** A bit array the same size as the number of windows in p_windows. Receives values indicating whether each window can be inserted. A set bit indicates the respective window cannot be inserted.

FB2K_MAKE_SERVICE_INTERFACE(*splitter_window_v2*, *splitter_window*)

4.2 Splitter items

class uie::**splitter_item_t**

Holds data about a splitter item.

Derive from here and also store your other stuff (show_caption..) Functions as data container only!

Subclassed by *uie::splitter_item_full_t*

Public Functions

virtual const GUID &**get_panel_guid**() const = 0

virtual void **set_panel_guid**(const GUID &p_guid) = 0

Setting GUID deletes panel config and window ptr (i.e. do it first)

virtual void **get_panel_config**(stream_writer *p_out) const = 0

virtual void **set_panel_config**(stream_reader *p_reader, t_size p_size) = 0

virtual const window_ptr &**get_window_ptr**() const = 0

inline virtual bool **query**(const GUID &p_guid) const

inline virtual **~splitter_item_t**()

template<typename **t_class**>
inline bool **query**(const *t_class* *&p_out) const

template<typename **t_class**>

inline bool **query**(*t_class* *&p_out)

inline void **get_panel_config_to_array**(pfc::array_t<uint8_t> &p_data, bool reset = false, bool refresh = false) const

inline pfc::array_t<uint8_t> **get_panel_config_to_array**(bool refresh = false) const

inline void **set_panel_config_from_ptr**(const void *p_data, t_size p_size)

template<class **t_base**>

class `uie::`**splitter_item_simple** : public *t_base*
Implements *splitter_item_t* with the standard set of data stored.

Public Functions

inline virtual const GUID &**get_panel_guid**() const

inline virtual void **get_panel_config**(stream_writer *p_out) const

inline virtual void **set_panel_guid**(const GUID &p_guid)

inline virtual void **set_panel_config**(stream_reader *p_reader, t_size p_size)

inline virtual const window_ptr &**get_window_ptr**() const

inline void **set_window_ptr**(const window_ptr &p_source)

class `uie::`**splitter_item_full_t** : public uie::*splitter_item_t*
Implements *splitter_item_t* with a full set of data stored.

Subclassed by *uie::splitter_item_full_v2_t*

Public Functions

virtual void **get_title**(pfc::string_base &p_out) const = 0

virtual void **set_title**(const char *p_title, t_size length) = 0

inline virtual bool **query**(const GUID &p_guid) const override

Public Members

uint32_t **m_caption_orientation** = { }

bool **m_locked** = { }

bool **m_hidden** = { }

bool **m_autohide** = { }

bool **m_show_caption** = { }

uint32_t **m_size** = { }

bool **m_show_toggle_area** = { }

bool **m_custom_title** = { }

Public Static Functions

static inline const GUID &**get_class_guid**()

class **uie::splitter_item_full_v2_t** : public uie::*splitter_item_full_t*
Subclassed by *uie::splitter_item_full_v3_t*

Public Functions

inline virtual bool **query**(const GUID &p_guid) const override

Public Members

uint32_t **m_size_v2** = { }

uint32_t **m_size_v2_dpi** = { }

Public Static Functions

static inline const GUID &**get_class_guid**()

class **uie::splitter_item_full_v3_t** : public uie::*splitter_item_full_v2_t*
Splitter item implementing support for additional data.

Use this when your splitter window needs to store additional data for each child panel that's not covered by the standard variables.

Note: You can use *splitter_item_full_v3_impl_t* rather than implementing this class. Alternatively, you can derive from splitter_item_full_v3_base_t.

Public Functions

virtual void **get_extra_data**(stream_writer *writer) const = 0
Gets the additional data associated with this splitter item.

Note: Check that *get_extra_data_format_id()* matches your format ID before calling this, as splitter items from other splitter windows may be inserted into your window.

Note: The data returned by this function may be serialised and passed between foobar2000 instances via the clipboard. And, at some point, you may find that you need to change the structure of the data. Make sure that your code handles such changes gracefully.

Parameters **writer** – Stream that receives the additional data.

virtual GUID **get_extra_data_format_id**() const = 0
Gets a GUID to identify the format of the data returned by *get_extra_data()*

Returns The format identifier

inline virtual bool **query**(const GUID &p_guid) const override

Public Static Functions

static inline const GUID &**get_class_guid**()

class **uie::splitter_item_full_v3_impl_t** : public
uie::splitter_item_full_impl_base_t<*splitter_item_full_v3_t*>
Implements *splitter_item_full_v3_t*.

Public Functions

inline void **get_extra_data**(stream_writer *writer) const override

inline GUID **get_extra_data_format_id**() const override

Public Members

pfc::array_t<t_uint8> **m_extra_data**

GUID **m_extra_data_format_id** = { }

CHAPTER

FIVE

WINDOW HELPERS

These classes can be used to make implementing panels easier.

5.1 Container window

class ui_helpers::**container_window**

Implements a window that serves either as an empty container for either other windows, or as a custom control.

Subclassed by ui_helpers::container_window_release_t, *uie::container_ui_extension_t< W, T >*

Public Functions

virtual *class_data* &**get_class_data**() const = 0

Gets window class data.

See also:

__implement_get_class_data, __implement_get_class_data_ex

Sample implementation:

```
virtual class_data & get_class_data() const
{
__implement_get_class_data(
"My Window Class", //window class name
true); //want transparent background (i.e. for toolbar controls)
}
```

Returns Reference to *class_data*

container_window()

HWND **create**(HWND wnd_parent, LPVOID create_param = 0, const ui_helpers::window_position_t &p_window_position = ui_helpers::window_position_null)

HWND **create_in_dialog_units**(HWND wnd_dialog, const ui_helpers::window_position_t &p_window_position, LPVOID create_param = NULL)

bool **ensure_class_registered**()

bool **class_release**()

void **destroy**()

HWND **get_wnd**() const

virtual LRESULT **on_message**(HWND wnd, UINT msg, WPARAM wp, LPARAM lp) = 0

Public Static Functions

static LRESULT WINAPI window_proc (HWND wnd, UINT msg, WPARAM wp, LPARAM lp)

struct **class_data**

Public Members

LPCTSTR **class_name**

LPCTSTR **window_title**

long **refcount**

bool **class_registered**

bool **want_transparent_background**

int **extra_wnd_bytes**

DWORD **styles**

DWORD **ex_styles**

UINT **class_styles**

bool **forward_system_settings_change**

bool **forward_system_colours_change**

bool **forward_system_time_change**

LPWSTR **cursor**

template<class **W** = ui_helpers::*container_window*, class **T** = *window*>

class `uie::container_ui_extension_t` : public ui_helpers::*container_window*, public uie::*window*

Wraps *ui_helpers::container_window* into a panel.

Public Functions

inline virtual HWND **create_or_transfer_window**(HWND parent, const window_host_ptr &host, const ui_helpers::window_position_t &p_position)

Create or transfer extension window.

Create your window here.

In the case of single instance panels, if your window is already created, you must (in the same order):

- Hide your window. i.e:

```
ShowWindow(wnd, SW_HIDE)
```

- Set the parent window to to wnd_parent. I.e.

```
SetParent(get_wnd(), wnd_parent)
```

- Move your window to the new window position. I.e.:

```
SetWindowPos(get_wnd(), NULL, p_position.x, p_position.y, p_position.cx, p_
↪position.cy, SWP_NOZORDER);
```

- Call relinquish_ownership() on your current host.

Other rules you should follow are:

- Ensure you are using the correct window styles. The window MUST have the WS_CHILD window style. It MUST NOT have the WS_POPUP, WS_CAPTION styles.
- The window must be created hidden.
- Use WS_EX_CONTROLPARENT if you have child windows that receive keyboard input, and you want them to be included in tab operations in the host window.
- Do not directly create a common control as your window. You must create a window to contain any common controls, and any other controls that communicate to the parent window via WM_COMMAND and WM_NOTIFY window messages.
- Under NO CIRCUMSTANCES may you subclass the host window.

- If you are not hosting any panels yourself, you may dialog manage your window if you wish.
- The window MUST have a dialog item ID of 0.

Parameters

- **wnd_parent** – **[in]** Handle to the window to use as the parent for your window
- **p_host** – **[in]** Pointer to the host that creates the extension. This parameter may not be NULL.
- **p_position** – **[in]** Initial position of the window

Pre May only be called if *is_available()* returned true.

Returns Window handle of the panel window

inline virtual void **destroy_window**()
Destroys the extension window.

inline virtual bool **is_available**(const window_host_ptr &p) const
Get availability of the extension.

This method is called before create_or_transfer() to test, if this call will be legal. If this instance is already hosted, it should check whether the given host's GUID equals its current host's GUID, and should return `false`, if it does. This is mostly important for single instance extensions.

Extensions that support multiple instances can generally return `true`.

Returns whether this instance can be created in or moved to the given host

inline const window_host_ptr &**get_host**() const

inline virtual HWND **get_wnd**() const
Gets extension window handle.

Pre May only be called on hosted extensions.

Returns Window handle of the extension window

inline virtual LPVOID **get_create_param**()

typedef *container_ui_extension_t*<ui_helpers::*container_window*, uie::*window*> **uie::container_ui_extension**

CHAPTER

SIX

VISUALISATION

These interfaces can be used to embed visualisations in other windows.

uie::visualisation is implemented by the built-in spectrum analyser visualisation.

6.1 Client

class **uie::visualisation** : public uie::*extension_base*

Interface for vis_extension service. This service allows you to embed the default Columns UI visualisation, and any other visualisations that implement it, into your own window.

Public Functions

virtual void **enable**(const visualisation_host_ptr &p_host) = 0

Enables the visualisation.

Parameters **p_host** – **[in]** Pointer to host to use for drawing operations

virtual void **paint_background**(HDC dc, const RECT *rc_area) = 0

Paints the standard background of your visualisation.

virtual void **disable**() = 0

Disables the visualisation.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*visualisation*)

Public Static Functions

static inline void **create_by_guid**(const GUID &guid, visualisation_ptr &p_out)

Create extension by GUID.

Parameters **guid** – **[in]** GUID of a vis_extension

6.2 Host

class uie::**visualisation_host** : public service_base
Interface for visualisation extension hosts.

Public Types

typedef pfc::refcounted_object_ptr_t<*painter_t*> **painter_ptr**

Public Functions

virtual void **create_painter**(*painter_ptr* &p_out) = 0
Creates a *painter_t* object.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*visualisation_host*)

class **painter_t** : public refcounted_object_root
Interface to paint on a visualistion host.

Note: Releasing the object ends the paint operation, frees the DC and updates the screen.

Public Functions

virtual HDC **get_device_context**() const = 0

virtual const RECT ***get_area**() const = 0

6.3 Factory

template<class **T**>

class **visualisation_factory** : public service_factory_t<*T*>
Service factory for vis extensions.

Usage example

```
static vis_extension_factory< my_vis_extension > foo_vis;
```

CHAPTER

SEVEN

CONTEXT MENU

These interfaces are used to implement context menu items for panels and toolbars.

7.1 Node

class `uie::menu_node_t` : public refcounted_object_root
Menu item interface class.

Remark

Remember, its derived from pfc::refcounted_object_root. So instantiate like:

```
uie::mode_node_ptr = new menu_node_impl;
```

Note: Do not derive directly from this; instead derive from either *menu_node_command_t*, *menu_node_popup_t* or *menu_node_separator_t*

Subclassed by *uie::menu_hook_impl*, *uie::menu_node_command_t*, *uie::menu_node_popup_t*, *uie::menu_node_separator_t*

Public Types

enum **state_t**
State of the menu item

Values:

enumerator **state_checked**

enumerator **state_disabled**

enumerator **state_greyed**

enumerator **state_disabled_grey**

enumerator **state_radio**

enumerator **state_radiochecked**

enum **type_t**
Type of the menu item

Values:

enumerator **type_popup**

enumerator **type_command**

enumerator **type_separator**

Public Functions

virtual *type_t* **get_type**() const = 0
Retrieves the type of the menu item.

Returns Type of the menu item.

virtual t_size **get_children_count**() const = 0
Retrieves the number of child items.

Pre May only return a non-zero value if your item is of type type_popup.

Returns Number of child items.

virtual void **get_child**(t_size index, menu_node_ptr &p_out) const = 0
Retrieves child item.

Parameters

- **index** – **[in]** Index of the child item to retrieve
- **p_out** – **[out]** Receives pointer to the child item

virtual bool **get_display_data**(pfc::string_base &p_out, unsigned &p_state) const = 0
Gets display data.

Parameters

- **p_out** – **[out]** Receives display text, utf-8 encoded. Valid only if flag_separator is not specified
- **p_state** – **[out]** Receives display state, combination of state_t flags.

Returns true iff the item should be displayed

virtual bool **get_description**(pfc::string_base &p_out) const = 0
Gets item description.

Parameters **p_out** – **[out]** Receives item description, utf-8 encoded.

Returns true iff the item has a description and p_out was set to it

virtual void **execute**() = 0
Executes the command. Applicable only for type_command items.

class **uie::menu_node_command_t** : public uie::*menu_node_t*
Base class for command menu items.

Subclassed by *uie::menu_node_configure*, *uie::simple_command_menu_node*

Public Functions

inline virtual type_t **get_type**() const
Retrieves the type of the menu item.

Returns Type of the menu item.

inline virtual t_size **get_children_count**() const
Retrieves the number of child items.

Pre May only return a non-zero value if your item is of type type_popup.

Returns Number of child items.

inline virtual void **get_child**(t_size index, menu_node_ptr &p_out) const
Retrieves child item.

Parameters

- **index** – **[in]** Index of the child item to retrieve
- **p_out** – **[out]** Receives pointer to the child item

class **uie::menu_node_popup_t** : public uie::*menu_node_t*
Base class for popup menu items.

Public Functions

inline virtual type_t **get_type**() const
Retrieves the type of the menu item.

Returns Type of the menu item.

inline virtual void **execute**()
Executes the command. Applicable only for type_command items.

inline virtual bool **get_description**(pfc::string_base &p_out) const
Gets item description.

Parameters **p_out** – **[out]** Receives item description, utf-8 encoded.

Returns true iff the item has a description and p_out was set to it

class **uie::menu_node_separator_t** : public uie::*menu_node_t*
Implements *menu_node_t* as a separator item.

Public Functions

inline virtual type_t **get_type**() const
Retrieves the type of the menu item.

Returns Type of the menu item.

inline virtual void **execute**()
Executes the command. Applicable only for type_command items.

inline virtual bool **get_description**(pfc::string_base &p_out) const
Gets item description.

Parameters **p_out** – **[out]** Receives item description, utf-8 encoded.

Returns true iff the item has a description and p_out was set to it

inline virtual t_size **get_children_count**() const
Retrieves the number of child items.

Pre May only return a non-zero value if your item is of type type_popup.

Returns Number of child items.

inline virtual bool **get_display_data**(pfc::string_base &p_out, unsigned &p_displayflags) const
Gets display data.

Parameters

- **p_out** – **[out]** Receives display text, utf-8 encoded. Valid only if flag_separator is not specified
- **p_state** – **[out]** Receives display state, combination of state_t flags.

Returns true iff the item should be displayed

inline virtual void **get_child**(t_size index, menu_node_ptr &p_out) const
Retrieves child item.

Parameters

- **index** – **[in]** Index of the child item to retrieve
- **p_out** – **[out]** Receives pointer to the child item

class uie::**simple_command_menu_node** : public uie::*menu_node_command_t*
Helper class to instantiate simple command menu nodes.

Public Functions

inline **simple_command_menu_node**(const char *display_name, const char *description, uint32_t display_flags, std::function<void()> on_execute)

inline virtual bool **get_display_data**(pfc::string_base &p_out, unsigned &p_displayflags) const override
Gets display data.

Parameters

- **p_out** – **[out]** Receives display text, utf-8 encoded. Valid only if flag_separator is not specified
- **p_state** – **[out]** Receives display state, combination of state_t flags.

Returns true iff the item should be displayed

inline virtual bool **get_description**(pfc::string_base &p_out) const override
Gets item description.

Parameters **p_out** – **[out]** Receives item description, utf-8 encoded.

Returns true iff the item has a description and p_out was set to it

inline virtual void **execute**() override
Executes the command. Applicable only for type_command items.

class uie::**menu_node_configure** : public uie::*menu_node_command_t*
Standard implementation of *uie::menu_node_command_t*, for an "Options" menu item.

Public Functions

inline virtual bool **get_display_data**(pfc::string_base &p_out, unsigned &p_displayflags) const
Gets display data.

Parameters

- **p_out** – **[out]** Receives display text, utf-8 encoded. Valid only if flag_separator is not specified
- **p_state** – **[out]** Receives display state, combination of state_t flags.

Returns true iff the item should be displayed

inline virtual bool **get_description**(pfc::string_base &p_out) const
Gets item description.

Parameters **p_out** – **[out]** Receives item description, utf-8 encoded.

Returns true iff the item has a description and p_out was set to it

inline virtual void **execute**()
Executes the command. Applicable only for type_command items.

inline **menu_node_configure**(*window* *wnd, const char *p_title = "Options")

7.2 Node receiver

class uie::**menu_hook_t**
Class that collects *menu_node_t* objects.

Subclassed by *uie::menu_hook_impl*

Public Functions

virtual void **add_node**(const menu_node_ptr &p_node) = 0

class **uie::menu_hook_impl** : public uie::*menu_hook_t*, public uie::*menu_node_t*
Standard implementation of *menu_hook_t*, also exposes *menu_node_t* interface.

Remark

Remember, its derived from pfc::refcounted_object_root. So instantiate like:

```
pfc::refcounted_ptr_t<uie::menu_hook_impl> = new uie::menu_hook_impl;
```

Public Functions

virtual void **add_node**(const menu_node_ptr &p_node)

virtual t_size **get_children_count**() const
Retrieves the number of child items.

Pre May only return a non-zero value if your item is of type type_popup.

Returns Number of child items.

virtual void **get_child**(t_size p_index, menu_node_ptr &p_out) const
Retrieves child item.

Parameters

- **index** – **[in]** Index of the child item to retrieve
- **p_out** – **[out]** Receives pointer to the child item

virtual type_t **get_type**() const
Retrieves the type of the menu item.

Returns Type of the menu item.

virtual bool **get_display_data**(pfc::string_base &p_out, unsigned &p_displayflags) const
Gets display data.

Parameters

- **p_out** – **[out]** Receives display text, utf-8 encoded. Valid only if flag_separator is not specified
- **p_state** – **[out]** Receives display state, combination of state_t flags.

Returns true iff the item should be displayed

virtual bool **get_description**(pfc::string_base &p_out) const
Gets item description.

Parameters **p_out** – **[out]** Receives item description, utf-8 encoded.

Returns true iff the item has a description and p_out was set to it

virtual void **execute**()
Executes the command. Applicable only for type_command items.

void **win32_build_menu**(HMENU menu, unsigned base_id, unsigned max_id)

void **execute_by_id**(unsigned id_exec)

CHAPTER

EIGHT

COLOURS

These interfaces are used to implement clients for centralised colour configuration.

namespace `cui::colours`

Enums

enum **colour_identifier_t**
Values:

enumerator **colour_text**

enumerator **colour_selection_text**

enumerator **colour_inactive_selection_text**

enumerator **colour_background**

enumerator **colour_selection_background**

enumerator **colour_inactive_selection_background**

enumerator **colour_active_item_frame**

enumerator **colour_group_foreground**
Reserved

enumerator **colour_group_background**
Reserved

enum **colour_flag_t**
Values:

enumerator **colour_flag_text**

enumerator **colour_flag_selection_text**

enumerator **colour_flag_inactive_selection_text**

enumerator **colour_flag_background**

enumerator **colour_flag_selection_background**

enumerator **colour_flag_inactive_selection_background**

enumerator **colour_flag_active_item_frame**

enumerator **colour_flag_group_foreground**

enumerator **colour_flag_group_background**

enumerator **colour_flag_all**

enum **bool_identifier_t**
Values:

enumerator **bool_use_custom_active_item_frame**

enumerator **bool_dark_mode_enabled**
Implemented in Columns UI 2.0. Always false on older versions.

See also:

helper for more details

enum **bool_flag_t**
Values:

enumerator **bool_flag_use_custom_active_item_frame**

enumerator **bool_flag_dark_mode_enabled**

enum **colour_mode_t**
Values:

enumerator **colour_mode_global**

enumerator **colour_mode_system**

enumerator **colour_mode_themed**

enumerator **colour_mode_custom**

Functions

static COLORREF **g_get_system_color**(const *colour_identifier_t* p_identifier)

bool **is_dark_mode_active**()
Get whether the UI-wide dark mode is currently active.

Convenience method to avoid having to instantiate a helper instance.

See also:

helper::is_dark_mode_active() for more details.

class **client** : public service_base
#include <columns_ui_appearance.h>

Public Functions

virtual const GUID &**get_client_guid**() const = 0

virtual void **get_name**(pfc::string_base &p_out) const = 0

inline virtual t_size **get_supported_colours**() const

virtual t_size **get_supported_bools**() const = 0
Return a combination of bool_flag_t to indicate which boolean flags are supported.

If dark mode is supported by your panel, you should set the *bool_flag_dark_mode_enabled* bit.

virtual bool **get_themes_supported**() const = 0
Indicates whether you are Theme API aware and can draw selected items using Theme API

virtual void **on_colour_changed**(t_size mask) const = 0

virtual void **on_bool_changed**(t_size mask) const = 0
Called whenever a supported boolean flag changes. Support for a flag is determined using the *get_supported_bools()* method.

Note: Only *bool_flag_dark_mode_enabled* is currently supported. Ensure you inspect mask to check which flags have changed.

Parameters **mask** – **[in]** a combination of bool_flag_t indicating the flags that have changed. (Only indicates which flags have changed, not the new values.)

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*client*)

template<class **tClass**>

class **factory** : public service_factory_t<*tClass*>
#include <columns_ui_appearance.h>

class **common_callback**
#include <columns_ui_appearance.h> Use this class if you wish to use the global colours only rather than implementing the client class

Subclassed by *cui::colours::dark_mode_notifier*

Public Functions

virtual void **on_colour_changed**(t_size mask) const = 0

virtual void **on_bool_changed**(t_size mask) const = 0

class **dark_mode_notifier** : private cui::*colours*::*common_callback*
#include <columns_ui_appearance.h> Helper for receiving notifications when the global dark mode status changes.

This is mainly used by non-panel parts of the UI. Panels would normally receive this notification through the on_bool_changed method of their client instance.

Public Functions

inline **dark_mode_notifier**(std::function<void()> callback)

inline **~dark_mode_notifier**()

inline virtual void **on_colour_changed**(t_size mask) const override

inline virtual void **on_bool_changed**(t_size mask) const override

class **helper**

#include <columns_ui_appearance.h> Helper to simplify retrieving colours.

Public Functions

inline COLORREF **get_colour**(const cui::*colours*::*colour_identifier_t* &p_identifier) const

inline bool **get_bool**(const cui::*colours*::*bool_identifier_t* &p_identifier) const

inline bool **get_themed**() const

inline bool **is_dark_mode_active**() const

Get whether the UI-wide dark mode is currently active.

Implemented in Columns UI 2.0. Always false on older versions.

There is only one global value of this flag; it does not vary between colour clients.

If your window contains a scroll bar, you should call SetWindowTheme based on the value of this flag as follows:

```
const auto dark_mode_active = cui::colours::is_dark_mode_active();
SetWindowTheme(wnd, dark_mode_active ? L"DarkMode_Explorer" : nullptr,␣
↪nullptr);
```

You should also do this when the *client::on_bool_changed()* method of your client is called with the *bool_flag_dark_mode_enabled* bit set.

inline **helper**(GUID guid = GUID{})

You can omit guid for the global colours

class **manager** : public service_base

#include <columns_ui_appearance.h> One implementation in Columns UI - do not reimplement!

It is not recommended to use this class directly - use the helper class instead.

Public Functions

virtual void **create_instance**(const GUID &p_client_guid, cui::*colours*::*manager_instance*::ptr &p_out) = 0

Creates a *manager_instance* for the given client (null GUID implies global settings).

inline virtual void **register_common_callback**(*common_callback* *p_callback)

inline virtual void **deregister_common_callback**(*common_callback* *p_callback)

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*manager*)

class **manager_instance** : public service_base

#include <*columns_ui_appearance.h*> One implementation in Columns UI - do not reimplement!

Public Functions

virtual COLORREF **get_colour**(const *colour_identifier_t* &p_identifier) const = 0

Get the specified colour.

virtual bool **get_bool**(const *bool_identifier_t* &p_identifier) const = 0

Get the specified colour.

virtual bool **get_themed**() const = 0

Only returns true if your *client::get_themes_supported()* method does. Indicates selected items should be drawn using Theme API.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*manager_instance*)

CHAPTER

NINE

FONTS

These interfaces are used to implement clients for centralised font configuration.

namespace `cui::fonts`

Enums

enum **font_mode_t**
Values:

enumerator **font_mode_common_items**

enumerator **font_mode_common_labels**

enumerator **font_mode_custom**

enumerator **font_mode_system**

enum **font_type_t**
Values:

enumerator **font_type_items**

enumerator **font_type_labels**

enum **font_type_flag_t**
Values:

enumerator **font_type_flag_items**

enumerator **font_type_flag_labels**

class **client** : public service_base
#include <columns_ui_appearance.h>

Public Functions

virtual const GUID &**get_client_guid**() const = 0

virtual void **get_name**(pfc::string_base &p_out) const = 0

virtual *font_type_t* **get_default_font_type**() const = 0

virtual void **on_font_changed**() const = 0

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*client*)

Public Static Functions

static bool **create_by_guid**(const GUID &p_guid, ptr &p_out)

template<class **tClass**>

class **factory** : public service_factory_t<*tClass*>
#include <columns_ui_appearance.h>

class **common_callback**
#include <columns_ui_appearance.h> Use this class if you wish to use the common fonts rather than implementing client

Public Functions

virtual void **on_font_changed**(t_size mask) const = 0

class **helper**
#include <columns_ui_appearance.h> Helper to simplify retrieving the font for a specific client.

Public Functions

inline void **get_font**(LOGFONT &p_out) const

inline HFONT **get_font**() const

inline **helper**(GUID p_guid)

class **manager** : public service_base
#include <columns_ui_appearance.h> One implementation in Columns UI - do not reimplement!

Public Functions

virtual void **get_font**(const GUID &p_guid, LOGFONT &p_out) const = 0
Retrieves the font for the given client.

virtual void **get_font**(const *font_type_t* p_type, LOGFONT &p_out) const = 0
Retrieves common fonts.

virtual void **set_font**(const GUID &p_guid, const LOGFONT &p_font) = 0
Sets your font as 'Custom' and to p_font.

virtual void **register_common_callback**(*common_callback* *p_callback) = 0

virtual void **deregister_common_callback**(*common_callback* *p_callback) = 0

inline HFONT **get_font**(const GUID &p_guid) const
Helper

inline HFONT **get_font**(const *font_type_t* p_type) const
Helper

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*manager*)

class **manager_v2** : public service_base
#include <columns_ui_appearance.h> Experimental version of the font management API with custom DPI support.

One implementation in Columns UI - do not reimplement!

Public Functions

virtual LOGFONT **get_client_font**(GUID guid, unsigned dpi = USER_DEFAULT_SCREEN_DPI) const = 0
Retrieve the font for the given client.

virtual LOGFONT **get_common_font**(*font_type_t* type, unsigned dpi = USER_DEFAULT_SCREEN_DPI) const = 0
Retrieve a common font.

virtual void **set_client_font**(GUID guid, const LOGFONT &font, int point_size_tenths) = 0
Set your font as 'Custom' and to the specified font.

virtual void **register_common_callback**(*common_callback* *callback) = 0

virtual void **deregister_common_callback**(*common_callback* *callback) = 0

inline HFONT **get_client_font_handle**(GUID guid, unsigned dpi =
USER_DEFAULT_SCREEN_DPI) const

inline HFONT **get_common_font_handle**(const *font_type_t* type, unsigned dpi =
USER_DEFAULT_SCREEN_DPI) const

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*manager_v2*)

CHAPTER

TEN

BUTTON

These interfaces are used to implement custom buttons for the Columns UI buttons toolbar.

10.1 Constants

enum `uie::t_button_guid`
Identifies the type of GUID.

Values:

enumerator **BUTTON_GUID_BUTTON**
GUID identifies a button command

enumerator **BUTTON_GUID_MENU_ITEM_CONTEXT**
GUID identifies a context menu command

enumerator **BUTTON_GUID_MENU_ITEM_MAIN**
GUID identifies a main menu command

enum `uie::t_button_type`
Identifies the type of button.

Values:

enumerator **BUTTON_TYPE_NORMAL**
The button acts as a standard click button

enumerator **BUTTON_TYPE_DROPDOWN**
The button displays a drop-down menu when pressed

enumerator **BUTTON_TYPE_DROPDOWN_ARROW**
The button displays an arrow which displays a drop-down menu

enum `uie::t_button_state`
Identifies the state of a button.

Combine multiple flags using bitwise or.

See also:

button::get_button_state

Values:

enumerator **BUTTON_STATE_ENABLED**
The button is enabled

enumerator **BUTTON_STATE_PRESSED**
The button is in an active state

enumerator **BUTTON_STATE_SHOW_TOOLTIP**
The button displays a ToolTip

enumerator **BUTTON_STATE_DEFAULT**
The default button state

enum `uie::t_mask`
Values:

enumerator **MASK_NONE**
No transparency mask is used.

enumerator **MASK_BITMAP**
1bpp bitmap transparency mask is used

enumerator **MASK_COLOUR**
Pixels with specified colour are transparent.

10.2 Interfaces

class `uie::button` : public service_base
Service that provides buttons for a toolbar.

Subclassed by *uie::button_v2*, *uie::custom_button*, *uie::menu_button*

Public Functions

virtual const GUID &**get_item_guid**() const = 0
Get the identifier of the button.

Use get_type_guid() to determine what the GUID repreesnts.

Returns GUID identifying the command represented by the class

inline virtual *t_button_guid* **get_guid_type**() const
Get whether *get_item_guid()* specifies a main menu item, a context menu, or a custom button command.

\Note Only recommended use of button-only buttons are dropdown-only buttons

See also:

t_button_guid

Returns type of command represnted by this class

virtual HBITMAP **get_item_bitmap**(unsigned command_state_index, COLORREF cr_btntext, *t_mask* &p_mask_type, COLORREF &cr_mask, HBITMAP &bm_mask) const = 0

Get a handle to a bitmap and its transparency mask of the menu item.

Deprecated:
Use *button_v2::get_item_bitmap()* instead.

Caller presumes ownership of bitmap.

Remark

Masks generated from a colour are only supported on bitmaps with a colour depth less than or equal to 8bpp.

Note: In the toolbar control, transparency masks are supported on all versions of windows; where as 32 bpp bitmaps with 8bpp alpha channel are supported only under common controls version 6.

Note: Ensure you do not create a mask bitmap if you fail to create main bitmap

---

Parameters

- **cr_btntext** – **[in]** Colour to use for text/foreground
- **bm_mask** – **[out]** HBITMAP of transparency mask. This is a monochrome bitmap.

Returns HBITMAP of menu item

inline virtual *t_button_type* **get_button_type**() const
Get type of button.

See also:

t_button_type

Returns Type of button

inline virtual void **get_menu_items**(*menu_hook_t* &p_out)
Gets menu items for drop-down buttons.

Parameters **p_out** – **[out]** Receives menu items

inline virtual unsigned **get_button_state**() const
Gets buttons state.

See also:

t_button_state

Returns Button state

inline virtual unsigned **get_command_state_index**() const
Gets current state of the command. For example, in a "Play or pause" command this would indicate the play or pause state.

Returns Index of current command state

inline virtual unsigned **get_command_state_count**() const
Gets total count of possible command states.

Returns Total count of possible command states

inline virtual void **get_command_state_name**(unsigned index, pfc::string_base &p_out) const
Gets name of specified command state.

Parameters

- **index** – **[in]** Index of command state's name to retrieve
- **p_out** – **[out]** Receives command state name

inline virtual void **register_callback**(*button_callback* &p_callback)
Registers a *button_callback* class to receive callbacks.

Parameters **p_callback** – **[in]** Reference to callback object requesting callbacks

inline virtual void **deregister_callback**(*button_callback* &p_callback)
Deregisters a *button_callback* class to stop receiving callbacks.

The object implementing this method must not keep any references to the specified callback object after this method returns

Parameters **p_callback** – **[in]** Reference to callback object being deregistered.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*button*)

class uie::**button_v2** : public uie::*button*
Extension of button interface; allows icons to be used as default button images.

New in SDK version 6.5.

Public Types

enum **handle_type_t**
Values:

enumerator **handle_type_bitmap**
HBITMAP

enumerator **handle_type_icon**
HICON

Public Functions

virtual HANDLE **get_item_bitmap**(unsigned command_state_index, COLORREF cr_btntext, unsigned cx_hint, unsigned cy_hint, unsigned &handle_type) const = 0
Get a handle to a image of the menu item.

Caller presumes ownership of bitmap.

Note: Use alpha channel for transparency.

Note: You can vary the returned image depending on whether dark mode is active by using *cui::colours::is_dark_mode_active()*. All button images are flushed when the dark mode status changes.

Parameters

- **command_state_index** – **[in]** Not used.
- **cr_btntext** – **[in]** Colour to use for text/foreground
- **cx_hint** – **[in]** Displayed bitmap width
- **cy_hint** – **[in]** Displayed bitmap height
- **handle_type** – **[out]** Receives the type of handle returned (icon or bitmap)

Returns Handle of image

inline virtual HBITMAP **get_item_bitmap**(unsigned command_state_index, COLORREF cr_btntext, *t_mask* &p_mask_type, COLORREF &cr_mask, HBITMAP &bm_mask) const
Not used.

FB2K_MAKE_SERVICE_INTERFACE(*button_v2*, *button*)

class **uie::menu_button** : public uie::*button*
Sub-class of *uie::button*, for buttons based upon a context menu item.

Public Functions

virtual void **select_subcommand**(const GUID &p_subcommand) = 0
Sets subcommand that subsequent function calls will refer to.

Called after instantiation, but before other command-related methods.

Parameters **p_subcommand** – **[in]** Specifies the subcommand that this object wll represent

FB2K_MAKE_SERVICE_INTERFACE(*menu_button*, *button*)

class **uie::custom_button** : public uie::*button*
Sub-class of *uie::button*, for buttons that implement their own command.

Public Functions

inline virtual *t_button_guid* **get_guid_type**() const
Get whether *get_item_guid()* specifies a main menu item, a context menu, or a custom button command.

\Note Only recommended use of button-only buttons are dropdown-only buttons

See also:

t_button_guid

Returns type of command represnted by this class

virtual void **execute**(const pfc::list_base_const_t<metadb_handle_ptr> &p_items) = 0
Executes the custom button's command.

Parameters **p_items** – **[in]** Items to perform the command on

virtual void **get_name**(pfc::string_base &p_out) const = 0
Gets the name of the custom button.

Parameters **p_out** – **[out]** Recieves the name of the button, UTF-8 encoded

inline virtual bool **get_description**(pfc::string_base &p_out) const
Gets the description of the custom button.

Parameters **p_out** – **[out]** Recieves the description of the button, UTF-8 encoded

Returns true iff the button has a description

FB2K_MAKE_SERVICE_INTERFACE(*custom_button*, *button*)

Public Static Functions

static inline bool **g_button_get_name**(const GUID &p_guid, pfc::string_base &p_out)

class uie::**button_callback**
Class implemented by button hosts to receive notification of button events.

Public Functions

virtual void **on_button_state_change**(unsigned p_new_state) = 0
Called when the state of the button changed

Parameters **p_new_state** – **[in]** Combination of *uie::t_button_state*

virtual void **on_command_state_change**(unsigned p_new_state) = 0
Called when the state of the command changed

See also:

button::get_command_state_index, *button::get_command_state_count*

Parameters **p_new_state** – **[in]** Index of new command state

10.3 Factories

template<class **T**>

class **button_factory** : public service_factory_t<*T*>
Service factory for buttons.

CHAPTER

ELEVEN

FCL FILES

class cui::fcl::**dataset** : public service_base
Subclassed by *cui::fcl::dataset_v2*

Public Functions

virtual void **get_name**(pfc::string_base &p_out) const = 0
User-friendly fully qualified (unambiguous) name.

virtual const GUID &**get_guid**() const = 0
Unique identifier of the dataset.

virtual const GUID &**get_group**() const = 0
The identifier of the group you belong to.

virtual void **get_data**(stream_writer *p_writer, t_uint32 type, *t_export_feedback* &feedback, abort_callback &p_abort) const = 0
Retrieves your data for an export.

Parameters **type** – **[in]** Specifies export mode. See t_fcl_type.

virtual void **set_data**(stream_reader *p_reader, t_size size, t_uint32 type, *t_import_feedback* &feedback, abort_callback &p_abort) = 0
Sets your data for an import.

Parameters **type** – **[in]** Specifies export mode. See t_fcl_type.

void **get_data_to_array**(pfc::array_t<uint8_t> &p_data, t_uint32 type, *t_export_feedback* &feedback, abort_callback &p_abort, bool b_reset = false) const
Helper function. Retrieves your data for an export.

See also:

get_data

Parameters **type** – **[in]** Specifies export mode. See t_fcl_type.

void **set_data_from_ptr**(const void *p_data, t_size size, t_uint32 type, *t_import_feedback* &feedback, abort_callback &p_abort)
Helper function. Sets your data for an import.

See also:

set_data

Parameters **type** – **[in]** Specifies export mode. See t_fcl_type.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*dataset*)

class cui::fcl::**dataset_v2** : public cui::fcl::*dataset*

Public Functions

inline virtual double **get_import_priority**() const

Determines the order in which data sets are imported when an FCL file is being imported.

New in Columns UI 1.1.

Data sets with a higher priority value are imported first.

This can be used when there are dependencies between global configuration data and panel instance data. Columns UI uses this internally to deprioritise the toolbar and layout data sets and you will not generally need to override this.

FB2K_MAKE_SERVICE_INTERFACE(*dataset_v2*, *dataset*)

template<class **T**>

class **dataset_factory** : public service_factory_single_t<*T*>

namespace cui::fcl::**groups**

Namespace containing standard FCL group identifiers.

Variables

constexpr GUID **layout** = {0x1979b677, 0x17ef, 0x4423, {0x94, 0x69, 0x11, 0x39, 0xa1, 0x1b, 0xd6, 0x87}}

constexpr GUID **toolbars** = {0xf1181b34, 0x8848, 0x43d0, {0x92, 0x96, 0x24, 0x48, 0x4c, 0x1f, 0x5b, 0xf1}}

constexpr GUID **colours_and_fonts** = {0xdd5158ae, 0xc8ed, 0x42d0, {0x89, 0xe3, 0xef, 0x1b, 0x19, 0x7f, 0xfc, 0xaf}}

constexpr GUID **title_scripts** = {0x2a8e63a4, 0xf8e, 0x459d, {0xb7, 0x52, 0x87, 0x4e, 0x38, 0x65, 0x8a, 0x6c}}

class cui::fcl::**group_impl_factory** : public service_factory_single_t<group_impl>
Helper.

Public Functions

inline **group_impl_factory**(const GUID &pguid, const char *pname, const char *pdesc, const GUID &pguidparent = pfc::guid_null)

class cui::fcl::**t_import_feedback**

Public Functions

virtual void **add_required_panel**(const char *name, const GUID &guid) = 0
Specifies any panels that you are dependent on that are not installed. You must specify only missing panels.

Parameters

- **name** – **[in]** Unused. Pass a null-terminiated empty string.
- **guid** – **[in]** GUID of panel.

class cui::fcl::**t_export_feedback**

Public Functions

virtual void **add_required_panels**(const pfc::list_base_const_t<GUID> &panels) = 0
Specifies panels that you are dependent on. You must specify all dependent panels.

Parameters **panels** – **[in]** GUIDs of panels.

inline void **add_required_panel**(GUID guid)

CHAPTER

TWELVE

WINDOW IDENTIFIERS

12.1 Built-in panels

namespace `cui::panels`

Namespace containing standard Columns UI panel GUIDs.

Variables

constexpr GUID **guid_playlist_switcher** = {0xc2cf9425, 0x540, 0x4579, {0xab, 0x3f, 0x13, 0xe2, 0x17, 0x66, 0x3d, 0x9b}}

constexpr GUID **guid_playlist_tabs** = {0xabb72d0d, 0xdbf0, 0x4bba, {0x8c, 0x68, 0x33, 0x57, 0xeb, 0xe0, 0x7a, 0x4d}}

constexpr GUID **guid_playlist_view**{0xf20bed8f, 0x225b, 0x46c3, {0x9f, 0xc7, 0x45, 0x4c, 0xed, 0xb6, 0xcd, 0xad}}

constexpr GUID **guid_vertical_splitter** = {0x77653a44, 0x66d1, 0x49e0, {0x9a, 0x7a, 0x1c, 0x71, 0x89, 0x8c, 0x4, 0x41}}

constexpr GUID **guid_horizontal_splitter** = {0x8fa0bc24, 0x882a, 0x4fff, {0x8a, 0x3b, 0x21, 0x5e, 0xa7, 0xfb, 0xd0, 0x7f}}

constexpr GUID **guid_filter** = {0xfb059406, 0xdddd, 0x4bd0, {0x8a, 0x11, 0x42, 0x42, 0x85, 0x4c, 0xbb, 0xa5}}

constexpr GUID **guid_artwork_view** = {0xdeead6ec, 0xf0b9, 0x4919, {0xb1, 0x6d, 0x28, 0xa, 0xed, 0xde, 0x73, 0x43}}

constexpr GUID **guid_playlist_view_v2** = {0xfb059406, 0x5f14, 0x4bd0, {0x8a, 0x11, 0x42, 0x42, 0x85, 0x4c, 0xbb, 0xa5}}

constexpr GUID **guid_item_details** = {0x59b4f428, 0x26a5, 0x4a51, {0x89, 0xe5, 0x39, 0x45, 0xd3, 0x27, 0xb4, 0xcb}}

constexpr GUID **guid_item_properties** = {0x8f6069cd, 0x2e36, 0x4ead, {0xb1, 0x71, 0x93, 0xf3, 0xdf, 0xf0, 0x7, 0x3a}}

12.2 Built-in toolbars

namespace `cui::toolbars`
Namespace containing standard Columns UI toolbar GUIDs.

Variables

constexpr GUID **guid_buttons** = {0xd8e65660, 0x64ed, 0x42e7, {0x85, 0xb, 0x31, 0xd8, 0x28, 0xc2, 0x52, 0x94}}

constexpr GUID **guid_menu** = {0x76e6db50, 0xde3, 0x4f30, {0xa7, 0xe4, 0x93, 0xfd, 0x62, 0x8b, 0x14, 0x1}}

constexpr GUID **guid_playback_order** = {0xaba09e7e, 0x9c95, 0x443e, {0xbd, 0xfc, 0x4, 0x9d, 0x66, 0xb3, 0x24, 0xa0}}

constexpr GUID **guid_spectrum_analyser** = {0xd947777c, 0x94c7, 0x409a, {0xb0, 0x2c, 0x9b, 0xe, 0xb9, 0xe3, 0x74, 0xfa}}

constexpr GUID **guid_seek_bar** = {0x678fe380, 0xabbb, 0x4c72, {0xa0, 0xb3, 0x72, 0xe7, 0x69, 0x67, 0x11, 0x25}}

constexpr GUID **guid_volume_control** = {0xb3259290, 0xcb68, 0x4d37, {0xb0, 0xf1, 0x80, 0x94, 0x86, 0x2a, 0x95, 0x24}}

constexpr GUID **guid_filter_search_bar** = {0x6e3b8b17, 0xaebd, 0x40d2, {0xa1, 0xf, 0x9d, 0x3a, 0xcf, 0x74, 0xf0, 0x91}}

12.3 Built-in visualisations

namespace `cui::visualisations`
Namespace containing standard Columns UI visualisation GUIDs.

Variables

constexpr GUID **`guid_spectrum_analyser`** = {0xd947777c, 0x94c7, 0x409a, {0xb0, 0x2c, 0x9b, 0xe, 0xb9, 0xe3, 0x74, 0xfa}}

CHAPTER

THIRTEEN

TITLE FORMATTING

template<bool **set** = true, bool **get** = true>

class cui::**titleformat_hook_global_variables** : public titleformat_hook

Public Functions

inline virtual bool **process_field**(titleformat_text_out *p_out, const char *p_name, unsigned p_name_length, bool &p_found_flag)

inline virtual bool **process_function**(titleformat_text_out *p_out, const char *p_name, unsigned p_name_length, titleformat_hook_function_params *p_params, bool &p_found_flag)

inline **titleformat_hook_global_variables**(*global_variable_list* &vars)

class cui::**global_variable_list** : public pfc::ptr_list_t<*global_variable*>

Public Functions

inline const char ***find_by_name**(const char *p_name, t_size length)

inline void **add_item**(const char *p_name, t_size p_name_length, const char *p_value, t_size p_value_length)

inline **~global_variable_list**()

class cui::**global_variable**

Public Functions

inline **global_variable**(const char *p_name, t_size p_name_length, const char *p_value, t_size p_value_length)

inline const char ***get_name**() const

inline const char ***get_value**() const

CHAPTER

FOURTEEN

SETTINGS

namespace cui::**config_objects**
Namespace containing Columns UI config_object GUIDs and related helper functions.

See also:

See config_object, config_object_notify and config_object_notify_impl_simple

Functions

inline bool **get_locked_panel_resizing_allowed**()
Gets whether resizing of locked panels should be allowed.

Remark

- In Columns UI 0.5.1 and older, this always returns true.

Returns Current value of 'Allow locked panel resizing' setting.

Variables

constexpr GUID **guid_bool_locked_panel_resizing_allowed**{0x3a0ef00a, 0xd538, 0x4470, {0x9a, 0x18, 0xdc, 0xf8, 0x22, 0xcc, 0x96, 0x73}}

namespace cui::**strings**
Namespace containing Columns UI string GUIDs.

Variables

constexpr GUID **guid_global_variables** = {0x493d419a, 0xcbb3, 0x4b8a, {0x8f, 0xb8, 0x28, 0xde, 0x2a, 0xe2, 0xf3, 0x6f}}

class `cui::control` : public service_base
Service exposing Columns UI control methods.

Remark

- One implementation in Columns UI, do not reimplement.
- Call from main thread only

Public Functions

virtual bool **get_string**(const GUID &p_guid, pfc::string_base &p_out) const = 0
Retrieves a string from Columns UI.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*control*)

CHAPTER

FIFTEEN

INDEX

- genindex

INDEX

C

U