
Columns UI SDK

Reupen Shah

Jan 26, 2023

GUIDES

1	Version 7.0.0-beta.2	3
1.1	New in this version	3
2	Version 7.0.0-beta.1	5
2.1	New in this version	5
2.2	Renamed namespaces	5
2.3	Deprecated in this version	6
2.4	Removed in this version	6
3	Dark mode	9
3.1	Handling dynamic dark mode changes	9
3.2	Painting panel backgrounds	10
3.3	Custom button images	11
4	Base extension	13
5	Window	17
5.1	Window	17
5.2	Playlist view	20
5.3	Menu	20
5.4	Factories	22
6	Splitter window	23
6.1	Splitter	23
6.2	Splitter items	26
7	Window helpers	31
7.1	Container window	31
7.2	Functions	34
8	Visualisation	35
8.1	Client	35
8.2	Host	36
8.3	Factory	36
9	Context menu	37
9.1	Node	37
9.2	Node receiver	41
10	Window host	43
10.1	Window host	43

10.2	Factories	47
11	Colours	49
12	Fonts	55
13	Button	59
13.1	Constants	59
13.2	Interfaces	60
13.3	Factories	65
14	FCL files	67
15	Window identifiers	71
15.1	Built-in panels	71
15.2	Built-in toolbars	72
15.3	Built-in visualisations	72
16	Title formatting	75
17	Settings	77
18	Getting started	79
18.1	Installation	79
18.2	Usage	79
18.3	Examples	79
18.4	Panel APIs	80
18.5	Button APIs	80
Index		81

VERSION 7.0.0-BETA.2

1.1 New in this version

This version adds an option to `uie::container_window_v3` to disable the forwarding of `WM_SETTINGCHANGE` messages to direct child windows.

This is useful when hosting the Win32 toolbar control as it can misbehave when handling this message.

VERSION 7.0.0-BETA.1

This version of the Columns UI SDK requires Visual Studio 2022.
The project file was also renamed `columns_ui-sdk-public.vcxproj`.

2.1 New in this version

This version adds:

- support for the Columns UI dark mode
- preliminary support for compiling x64 panels
- `uie::container_uie_window_v3_t`
- `uie::container_window_v3`
- `uie::simple_command_menu_node`
- `uie::splitter_item_full_v2_t`
- `uie::splitter_item_full_v3_t`
- `uie::splitter_item_full_v3_impl_t`
- `uie::splitter_window::size_and_dpi`
- `cui::fcl::dataset_v2`
- `cui::fonts::manager_v2`
- `cui::config_objects::get_locked_panel_resizing_allowed()`
- `uie::win32::paint_background_using_parent()`

2.2 Renamed namespaces

- The `ui_extension` namespace was renamed `uie`
- The `columns_ui` namespace was renamed `cui`

Aliases exist for the old names for backwards compatibility.

2.3 Deprecated in this version

The following classes were deprecated:

- `uie::container_ui_extension_t`
- `uie::container_ui_extension`
- `uie::container_menu_ui_extension`
- `uie::container_uie_window_t`
- `uie::container_window_autorelease_t`
- `uie::container_window_release_t`
- `uie::container_window`

The following functions were deprecated:

- `uHeader_InsertItem()`
- `uHeader_SetItemText()`
- `uHeader_SetItemWidth()`
- `uToolTip_AddTool()`
- `uComboBox_SelectString()`
- `win32_helpers::send_message_to_all_children()`
- `win32_helpers::tooltip_add_tool()`

2.4 Removed in this version

Example components are no longer bundled with the SDK. These are now published on GitHub:

- [Example panel](#)
- [Console panel](#)

The following classes were removed:

- `uie::window_base_t`
- `logfont_os_menu`
- `logfont_os_icon`
- `logfont_os_from_utf8`
- `logfont_utf8_from_os`

The following functions were removed:

- `uGetClassLong()`
- `uSetClassLong()`
- `convert_logfont_utf8_to_os()`
- `convert_logfont_os_to_utf8()`

The following macros were removed:

- `uT()`

- `uTS()`
- `Tu()`
- `TSu()`

DARK MODE

Columns UI 2.0.0 and newer feature an optional dark mode on Windows 10 version 2004 and newer.

If dark mode is active, panels should render system UI elements, such as common controls and scroll bars, with a dark theme.

The following code sample shows how to enable dark scroll bars for a window depending on whether dark mode is enabled:

```
const auto is_dark = cui::colours::is_dark_mode_active()  
SetWindowTheme(hwnd, is_dark ? L"DarkMode_Explorer" : nullptr, nullptr);
```

Some common controls have a native dark mode that can also be activated using `SetWindowTheme()` and one of the following themes:

- `DarkMode`
- `DarkMode_Explorer`
- `DarkMode_CFD`

3.1 Handling dynamic dark mode changes

If you have an existing `cui::colours::client` implementation, you should:

1. Override `cui::colours::client::get_supported_bools()` to return the `cui::colours::bool_flag_dark_mode_enabled` flag (use `|` to combine multiple flags). For example:

```
uint32_t get_supported_bools() const override  
{  
    return colours::bool_flag_use_custom_active_item_frame | colours::bool_  
↳flag_dark_mode_enabled;  
}
```

2. Override `cui::colours::client::on_bool_changed()` to handle dynamic dark mode changes. For example:

```
void on_bool_changed(uint32_t changed_items_mask) const override  
{  
    if (changed_items_mask & colours::bool_flag_dark_mode_enabled) {  
        const auto is_dark = cui::colours::is_dark_mode_active();  
        for (auto hwnd : hwnds) {
```

(continues on next page)

(continued from previous page)

```

        SetWindowTheme(hwnd, is_dark ? L"DarkMode_Explorer" : nullptr,
↳ nullptr);
    }
}
}

```

If you don't have an existing `cui::colours::client` implementation, you can use `cui::colours::dark_mode_notifier` to react to dark mode status changes. For example:

```

// Member variable
std::unique_ptr<cui::colours::dark_mode_notifier> m_dark_mode_notifier;

// Window procedure
case WM_CREATE:
    // ...
    SetWindowTheme(hwnd, is_dark ? L"DarkMode_Explorer" : nullptr, nullptr);
    m_dark_mode_notifier = std::make_unique<cui::colours::dark_mode_notifier>
↳ ([hwnd] {
        SetWindowTheme(hwnd, is_dark ? L"DarkMode_Explorer" : nullptr,
↳ nullptr);
    });
    return 0;
case WM_DESTROY:
    m_dark_mode_notifier.reset();
    return 0;

```

3.2 Painting panel backgrounds

While many panels will automatically use the background colour configured in Columns UI, there are some additional considerations to avoid glitches e.g. when resizing panels.

If your panel uses a custom window class, it's recommended to set the `hbrBackground` member of the `WNDCLASS` structure to `nullptr` when registering the window class, and to explicitly handle erasing of your window's background. This is to avoid a non-dark system colour temporarily showing through when resizing panels with dark mode enabled.

If your entire client area is covered by e.g. a child common control, you can use the `uie::win32::paint_background_using_parent()` function to simply paint the parent window's background in the window procedure for your custom window class.

Similarly, if:

- you're currently using `uie::container_ui_extension` (or any of its related variants)
- `want_transparent_background` is set to `false` in your and `container_window::get_class_data()` implementation; and
- your window procedure doesn't handle `WM_ERASEBKGRND` explicitly

your panel is using `COLOR_BTNFACE` for its background (which is the same colour in both light and dark modes). If this applies, you should be able to migrate to `uie::container_uie_window_v3_t` with a transparent background to avoid `COLOR_BTNFACE` showing through when resizing panels.

3.3 Custom button images

If you have an implementation of `uie::button_v2`, you should generally make `uie::button_v2::get_item_bitmap()` vary the image returned according to the value returned by `cui::colours::is_dark_mode_active()`, so that a dark version of the image is returned when dark mode is active.

BASE EXTENSION

class **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 configuration 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

pf::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 – pf::exception on failure

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

void **export_config_to_array**(pf::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 – pf::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

WINDOW

These interfaces are used to implement panels and toolbars.

5.1 Window

```
class window : public uie::extension_base
```

Interface for window service.

Subclassed by `uie::container_ui_extension_t< W, T >`, `uie::container_uie_window_v3_t< Base >`, `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 whether 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.

5.2 Playlist view

class **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*)

5.3 Menu

class **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_GETKEYBOARDUCUES 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_GETKEYBOARDUCUES system parameter.

Remark

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

Implementation example

```
BOOL b_showkeyboardcues = TRUE;
SystemParametersInfo(SPI_GETKEYBOARDUCUES, 0, &b_showkeyboardcues, S_
↳0);
PostMessage(wnd_menu, WM_UPDATEUISTATE, MAKEWPARAM(b_
↳showkeyboardcues ? UIS_CLEAR : UIS_SET , UISF_HIDEACCEL),
0);
```

FB2K_MAKE_SERVICE_INTERFACE(*menu_window*, *window*)

class **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*)

5.4 Factories

```
template<class T>
```

```
class 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 void instance_create(service_ptr_t<service_base> &p_out) override
```

SPLITTER WINDOW

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

6.1 Splitter

class **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 ref-counted 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 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)
```

6.2 Splitter items

class **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 **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 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 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 **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 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

```
pf::array_t<t_uint8> m_extra_data
```

```
GUID m_extra_data_format_id = {}
```


WINDOW HELPERS

These classes can be used to make implementing panels easier.

7.1 Container window

struct **container_window_v3_config**

Window and window class styles, names and other parameters for a *container_window_v3*.

Public Functions

```
inline container_window_v3_config(const wchar_t *class_name, bool  
                                use_transparent_background = true, unsigned  
                                class_styles = 0)
```

Public Members

```
const wchar_t *class_name = {}
```

```
bool use_transparent_background = {true}
```

Whether to use the parent window's background for this window.

If true, `on_message()` will not be called when the `WM_ERASEBKGND` message is received. The window (but not its children) will also be invalidated on resize or move.

You can also set this to false and use `uie::win32::paint_background_using_parent()` in your `on_message()` implementation for more flexibility.

If set to false, you should ensure a background is painted for this window.

```
bool invalidate_children_on_move_or_resize = {}
```

```
bool forward_wm_settingchange = {true}
```

Whether to forward `WM_SETTINGCHANGE` messages to direct child windows.

This should be set to false if a toolbar control is a direct child window, as they can misbehave when handling `WM_SETTINGCHANGE`.

```
unsigned window_styles = {WS_CHILD | WS_CLIPCHILDREN | WS_CLIPSIBLINGS}
```

```
unsigned extended_window_styles = {WS_EX_CONTROLPARENT}
```

```
unsigned class_styles = {}
```

```
LPWSTR class_cursor = {IDC_ARROW}
```

```
HBRUSH class_background = {}
```

```
const wchar_t * window_title = {L""}
```

```
int class_extra_wnd_bytes = {}
```

```
class container_window_v3
```

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

Public Functions

```
inline container_window_v3(container_window_v3_config config,  
                           std::function<LRESULT(HWND wnd, UINT msg, WPARAM wp,  
                           LPARAM lp)> on_message = nullptr)
```

```
container_window_v3(const container_window_v3 &p_source) = delete
```

```
container_window_v3 &operator=(const container_window_v3 &p_source) = delete
```

```
HWND create(HWND wnd_parent, int x, int y, int cx, int cy)
```

```
HWND create(HWND wnd_parent)
```

```
void destroy() const
```

Destroy the window.

If this is the last instance of this window class, the window class will also be deregistered.

```
inline HWND get_wnd() const
```

```
void deregister_class() const
```

Deregister the window class.

If not using *destroy()* to destroy the window, call this to deregister the window class when all windows belonging to the class have been destroyed.

```
template<class Base = window>
```

```
class container_uie_window_v3_t : public uie::window
```

A base implementation of *uie::window* using *uie::container_window_v3*

Public Functions

virtual *container_window_v3_config* **get_window_config**() = 0

Get window and window class styles, names and other parameters.

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

inline virtual bool **is_available**(const window_host_ptr &p) const override

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 final

Gets extension window handle.

Pre May only be called on hosted extensions.

Returns Window handle of the extension window

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

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_
    istance.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()** final

Destroys the extension window.

using uie::container_uie_window_v3 = *container_uie_window_v3_t*<>

7.2 Functions

LRESULT uie::win32::**paint_background_using_parent**(HWND wnd, HDC dc, bool use_wm_printclient)

VISUALISATION

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

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

8.1 Client

class **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`

8.2 Host

class **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 visualisation 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
```

8.3 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;
```


CONTEXT MENU

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

9.1 Node

class **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 **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 override

Retrieves the type of the menu item.

Returns Type of the menu item.

inline virtual t_size **get_children_count**() const override

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 override

Retrieves child item.

Parameters

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

class **menu_node_popup_t** : public uie::menu_node_t

Base class for popup menu items.

Public Functions

inline virtual type_t **get_type**() const override

Retrieves the type of the menu item.

Returns Type of the menu item.

inline virtual void **execute**() override

Executes the command. Applicable only for type_command items.

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

class **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 override

Retrieves the type of the menu item.

Returns Type of the menu item.

inline virtual void **execute**() override

Executes the command. Applicable only for type_command items.

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 t_size **get_children_count**() const override

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 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 void **get_child**(t_size index, menu_node_ptr &p_out) const override

Retrieves child item.

Parameters

- **index** – [in] Index of the child item to retrieve

- **p_out** – [out] Receives pointer to the child item

class **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 **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 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.

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

9.2 Node receiver

class **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 **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) override

virtual t_size **get_children_count**() const override

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 override

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 override

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 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

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

virtual void **execute**() override

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)

WINDOW HOST

These interfaces are used by components that can host other panels and toolbars.

10.1 Window host

class **window_host** : public service_base

Interface for *window_host* service.

This interface is to be implemented by panel hosts.

Remark

The host may not be dialog managed.

Remark

Hosts must forward the following messages to hosted windows:

- WM_SETTINGCHANGE
 - WM_SYSCOLORCHANGE
-

Subclassed by *uie::window_host_ex*, *uie::window_host_with_control*

Public Functions

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

Get the unique ID of the host.

This GUID is used to identify a specific host.

Returns host GUID

virtual void **on_size_limit_change**(HWND wnd, unsigned flags) = 0

Notify host about changed size limits of a hosted extension.

See also:

uie::size_limit_flag_t

Parameters

- **wnd** – [in] window handle of the extension’s window
- **flags** – [in] a combination of SLC_* flags indicating which size limits changed

Pre May only be called by a hosted UI extension.virtual unsigned **is_resize_supported**(HWND wnd) const = 0

Called by panels hosted by this host to find out whether the host supports resizing.

See also:

uie::resize_flag_t

Parameters **wnd** – [in] handle to the window to test**Returns** combination of uie::size_height and uie::size_width to indicate whether the width or height can be modified**Pre** May only be called by a hosted UI extension.virtual bool **request_resize**(HWND wnd, unsigned flags, unsigned width, unsigned height) = 0

Called by ui extension hosted by this host to resize your window.

Implementers: If you cannot fully meet the request, do not attempt to partially fulfil it. For example, if a request is made to modify both the width and height but you can only modify one if those.

See also:

uie::resize_flag_t

Parameters **wnd** – [in] handle to the window to test**Returns** combination of uie::size_height and uie::size_width to indicate whether the width or height is being modifiedvirtual bool **override_status_text_create**(service_ptr_t<ui_status_text_override> &p_out) = 0

Instantiates ui_status_text_override service, that can be used to display status messages.

Implementers: if you wish to display status bar text in the main window, simply use ui_control::override_status_text_create. Hybrid panel-hosts can forward the call to their host. If alternatively you wish to display the text in your own status area, you are responsible for implementing ui_status_text_override. Be sure to obey certain conventions:

- Releasing the ui_status_text_override object should restore the text if revert_text has not been called.

Parameters **p_out** – [out] receives new ui_status_text_override instance.**Pre** May only be called by a hosted UI extension.**Returns** true on success, false on failure (out of memory / no GUI loaded / etc)inline virtual bool **get_keyboard_shortcuts_enabled**() const

Query if keyboard shortcuts should be processed.

Use this to determine, if keyboard shortcuts should be processed. Do not process them, if this method returns false. Shortcuts can be processed using the keyboard_shortcut_manager service from the foobar2000 SDK.

Keyboard shortcuts would not be processed, for example, if the panel is hosted in a popup window. In this case the method returns false.

If the method does return true, whether you process keyboard shortcuts will depend on the type of functionality your control offers. For example, in a edit control you may wish not to process keyboard shortcuts.

The user must be able to navigate using the tab key. If VK_TAB is not processed by the keyboard_shortcut_manager and the TAB press is not being handled by the dialog manager, you should use g_on_tab() to change to the next control.

Usage example

```

case WM_KEYDOWN:
    if (p_host->get_keyboardshortcuts_enabled() &&
static_api_ptr_t<keyboard_shortcut_manager>->on_keydown_xxxx(wp))
    break; else if (wp == VK_TAB)
window::g_on_tab(wnd); break;

```

Pre May only be called by a hosted UI extension.

Returns whether keyboard shortcuts should be processed

virtual bool **is_visible**(HWND wnd) const = 0

Query if extension window is visible.

An extension that is not visible does not imply that its window has been hidden using ShowWindow

Parameters **wnd** – [in] handle to the window to test

Pre May only be called by a hosted UI extension.

Returns whether window is visible.

virtual bool **is_visibility_modifiable**(HWND wnd, bool desired_visibility) const = 0

Query if extension window can be hidden or shown.

Parameters

- **wnd** – [in] handle to the window to test
- **desired_visibility** – [in] whether you want the window to be visible

Pre May only be called by a hosted UI extension.

Returns whether the required visibility can be set.

virtual bool **set_window_visibility**(HWND wnd, bool visibility) = 0

Hides or shows extension window.

Parameters

- **wnd** – [in] handle to the window to test
- **visibility** – [in] whether you want the window to be visible

Pre May only be called by a hosted UI extension.

Returns whether the required visibility was be set.

virtual void **relinquish_ownership**(HWND wnd) = 0

Relinquish ownership of a UI extension instance.

Call this to remove control of an extension window from the host. The host will not destroy the window as a result of this call. However, the window may be destroyed, if the host destroys the containing window, so be sure to call SetParent first.

Reasons for calling this method include: another host tries to take ownership of an existing extension instance, the window should be destroyed/closed, or the window is to be turned into a popup dialog.

See also:

window::create_or_transfer_window

Parameters *wnd* – [in] window handle of the extension’s window

Pre May only be called by a hosted UI extension.

FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT(*window_host*)

class **window_host_with_control** : public uie::*window_host*

Sub-class of *window_host*, providing methods for external control.

In addition to the methods exposed through the *window_host* interface, this interface provides information about the host and its state as well as methods to manage hosted extensions.

Public Functions

virtual unsigned **get_supported_types**() const = 0

Get supported UI extension types.

See also:

window_flag::window_type

Returns a combination of *window_flag::TYPE_** flags to indicate recommended types for the host

virtual void **insert_extension**(const GUID &guid, unsigned height, unsigned width) = 0

Insert new instance of a UI extension.

Creates an instance of the specified extension and inserts it into the host’s client area. Single-instance extensions should removed themselves from the old host, if any.

See also:

is_available, *window::init_or_take_ownership*

Parameters

- **guid** – [in] unique ID of the UI extension to be inserted
- **height** – [in] desired height of the new panel
- **width** – [in] desired width of the new panel

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

virtual void **insert_extension**(*window_ptr* &p_ext, unsigned height, unsigned width) = 0

Insert existing instance of a UI extension.

Inserts the given UI extension instance into the host’s client area.

See also:

is_available, *window::init_or_take_ownership*

Parameters

- **p_ext** – [in] pointer to the UI extension instance to be inserted

- **height** – [in] desired height of the new panel
 - **width** – [in] desired width of the new panel
- Pre** May only be called, if *is_available()* returned true.

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

Get the name of the host.

Get a user-readable name of the host.

See also:

get_host_guid

Warning: Do not use the name to identify hosts; use host GUIDs instead.

Parameters **out** – [out] receives the name of the host, e.g. “My UI/Sidebar”

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

Get availability of the host.

See also:

insert_extension(const GUID &, unsigned, unsigned), *insert_extension(window *, unsigned, unsigned)*

Returns true if it is possible to insert a UI extension into the host.

FB2K_MAKE_SERVICE_INTERFACE(*window_host_with_control*, *window_host*)

10.2 Factories

```
template<class T>
```

```
class window_host_factory : public service_factory_t<T>
```

Service factory for window hosts.

Usage example

```
static window_host_factory< my_window_host > foo_host;
```

```
template<class T>
```

```
class window_host_factory_single : public service_factory_single_t<T>
```

Service factory for window hosts.

Usage example

```
static window_host_factory< my_window_host > foo_host;
```

The static instance of `my_window_host` can be accessed as `foo_host.get_static_instance()`.

Public Functions

```
inline operator uie::window_host_ptr()
```

```
template<class T>
```

```
class window_host_factory_transparent_single : public  
service_factory_single_transparent_t<T>
```

Service factory for window hosts.

Usage example

```
static window_host_factory_transparent< my_window_host > foo_host2;
```

The static instance of `my_window_host` can be accessed as `foo_host2`.

COLOURS

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

namespace **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`

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

Public Functions

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

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

inline virtual uint32_t **get_supported_colours**() const

virtual uint32_t **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**(uint32_t changed_items_mask) const = 0

virtual void **on_bool_changed**(uint32_t changed_items_mask) const = 0

Called whenever a supported boolean flag changes. Support for a flag is determined using the *get_supported_bools()* method.

Example implementation:

```
void on_bool_changed(uint32_t changed_items_mask) const override
{
    if (changed_items_mask & colours::bool_flag_dark_mode_enabled) {
        const auto is_dark = cui::colours::is_dark_mode_active();
        // Handle dark mode change
    }
}
```

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

Parameters `changed_items_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>
```

```
class common_callback
```

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(uint32_t changed_items_mask) const = 0
```

```
virtual void on_bool_changed(uint32_t changed_items_mask) const = 0
```

```
class dark_mode_notifier : private cui::colours::common_callback
```

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(uint32_t changed_items_mask) const override
```

```
inline virtual void on_bool_changed(uint32_t changed_items_mask) const override
```

```
class helper
```

Helper to simplify retrieving colours.

Public Functions

```
inline COLORREF get_colour(const colour_identifier_t &p_identifier) const
```

```
inline bool get_bool(const 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" : L"
↳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
```

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
```

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)
```


FONTS

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

namespace **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

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>
```

```
class common_callback
```

Use this class if you wish to use the common fonts rather than implementing client

Public Functions

```
virtual void on_font_changed(uint32_t changed_items_mask) const = 0
```

```
class helper
```

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
```

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

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)

```


BUTTON

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

13.1 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.

13.2 Interfaces

class **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 represents.**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 represented 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 **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 override

Deprecated *uie::button* method, not used for *uie::button_v2*.

FB2K_MAKE_SERVICE_INTERFACE(*button_v2*, *button*)

class **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 will represent

FB2K_MAKE_SERVICE_INTERFACE(*menu_button*, *button*)

class **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 override

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 represented 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] Receives 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] Receives 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 **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

13.3 Factories

```
template<class T>
```

```
class button_factory : public service_factory_t<T>
```

```
    Service factory for buttons.
```


class **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 **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 **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 **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 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-terminated empty string.
- **guid** – [in] GUID of panel.

```
class 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)
```


WINDOW IDENTIFIERS

15.1 Built-in panels

namespace **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 = {0xab72d0d, 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}}
```

15.2 Built-in toolbars

namespace **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}}
```

15.3 Built-in visualisations

namespace **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}}
```


TITLE FORMATTING

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

```
class titleformat_hook_global_variables : public titleformat_hook
```

Public Functions

```
inline bool process_field(titleformat_text_out *p_out, const char *p_name, size_t  
p_name_length, bool &p_found_flag) override
```

```
inline bool process_function(titleformat_text_out *p_out, const char *p_name, size_t  
p_name_length, titleformat_hook_function_params *p_params,  
bool &p_found_flag) override
```

```
inline titleformat_hook_global_variables(global_variable_list &vars)
```

```
class 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 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
```


SETTINGS

namespace **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 **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 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)
```

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

18.1 Installation

You'll need:

- [Microsoft Visual Studio 2022](#)
- [foobar2000 SDK](#)

To install, [download the SDK](#) and extract the archive alongside the foobar2000 subdirectory of your foobar2000 SDK.

18.2 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.

18.3 Examples

Two examples are published on GitHub:

- [Example panel](#) – a simple panel that displays some text and implements a context menu item
- [Console panel](#) – a console viewer

18.4 Panel APIs

18.4.1 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.

18.4.2 Helpers

The preferred method of implementing the window class is to derive from `uie::container_uie_window_v3` (although this may not be suitable for single-instance panels or dialog-based panels).

18.5 Button APIs

18.5.1 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`.

18.5.2 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.

INDEX

C

`cui::colours` (C++ type), 49

`cui::colours::bool_flag_t` (C++ enum), 50

`cui::colours::bool_flag_t::bool_flag_dark_mode_enabled` (C++ enumerator), 50

`cui::colours::bool_flag_t::bool_flag_use_custom_active_item_frame` (C++ enumerator), 50

`cui::colours::bool_identifier_t` (C++ enum), 50

`cui::colours::bool_identifier_t::bool_dark_mode_enabled` (C++ enumerator), 50

`cui::colours::bool_identifier_t::bool_use_custom_active_item_frame` (C++ enumerator), 50

`cui::colours::client` (C++ class), 51

`cui::colours::client::factory` (C++ class), 52

`cui::colours::client::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT` (C++ function), 52

`cui::colours::client::get_client_guid` (C++ function), 51

`cui::colours::client::get_name` (C++ function), 51

`cui::colours::client::get_supported_bools` (C++ function), 51

`cui::colours::client::get_supported_colours` (C++ function), 51

`cui::colours::client::get_themes_supported` (C++ function), 51

`cui::colours::client::on_bool_changed` (C++ function), 51

`cui::colours::client::on_colour_changed` (C++ function), 51

`cui::colours::colour_flag_t` (C++ enum), 49

`cui::colours::colour_flag_t::colour_flag_active_item_frame` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_all` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_background` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_group_background` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_group_foreground` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_inactive_selection_background` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_inactive_selection` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_selection_background` (C++ enumerator), 50

`cui::colours::colour_flag_t::colour_flag_selection_text` (C++ enumerator), 49

`cui::colours::colour_flag_t::colour_flag_text` (C++ enumerator), 49

`cui::colours::colour_identifier_t` (C++ enum), 49

`cui::colours::colour_identifier_t::colour_active_item_frame` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_background` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_group_background` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_group_foreground` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_inactive_selection` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_inactive_selection_text` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_selection_background` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_selection_text` (C++ enumerator), 49

`cui::colours::colour_identifier_t::colour_text` (C++ enumerator), 49

`cui::colours::common_callback` (C++ class), 52

`cui::colours::common_callback::on_bool_changed` (C++ function), 52

`cui::colours::common_callback::on_colour_changed` (C++ function), 52

`cui::colours::dark_mode_notifier` (C++ class), 52

`cui::colours::dark_mode_notifier::~dark_mode_notifier` (C++ function), 52

`cui::colours::dark_mode_notifier::dark_mode_notifier` (C++ function), 52

`cui::colours::dark_mode_notifier::on_bool_changed` (C++ function), 52

cui::colours::dark_mode_notifier::on_colour_changed (function), 67
 (C++ function), 52
 cui::colours::g_get_system_color (C++ function), 51
 cui::colours::helper (C++ class), 52
 cui::colours::helper::get_bool (C++ function), 52
 cui::colours::helper::get_colour (C++ function), 52
 cui::colours::helper::get_themed (C++ function), 52
 cui::colours::helper::helper (C++ function), 53
 cui::colours::helper::is_dark_mode_active (C++ function), 52
 cui::colours::is_dark_mode_active (C++ function), 51
 cui::colours::manager (C++ class), 53
 cui::colours::manager::create_instance (C++ function), 53
 cui::colours::manager::deregister_common_callback (C++ function), 53
 cui::colours::manager::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 53
 cui::colours::manager::register_common_callback (C++ function), 53
 cui::colours::manager_instance (C++ class), 53
 cui::colours::manager_instance::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 53
 cui::colours::manager_instance::get_bool (C++ function), 53
 cui::colours::manager_instance::get_colour (C++ function), 53
 cui::colours::manager_instance::get_themed (C++ function), 53
 cui::config_objects (C++ type), 77
 cui::config_objects::get_locked_panel_resizing_allowed (C++ function), 77
 cui::config_objects::guid_bool_locked_panel_resizing_allowed (C++ member), 77
 cui::control (C++ class), 78
 cui::control::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 78
 cui::control::get_string (C++ function), 78
 cui::fcl::dataset (C++ class), 67
 cui::fcl::dataset::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 68
 cui::fcl::dataset::get_data (C++ function), 67
 cui::fcl::dataset::get_data_to_array (C++ function), 67
 cui::fcl::dataset::get_group (C++ function), 67
 cui::fcl::dataset::get_guid (C++ function), 67
 cui::fcl::dataset::get_name (C++ function), 67
 cui::fcl::dataset::set_data (C++ function), 67
 cui::fcl::dataset::set_data_from_ptr (C++ function), 67
 cui::fcl::dataset_factory (C++ class), 68
 cui::fcl::dataset_v2 (C++ class), 68
 cui::fcl::dataset_v2::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 68
 cui::fcl::dataset_v2::get_import_priority (C++ function), 68
 cui::fcl::group_impl_factory (C++ class), 68
 cui::fcl::group_impl_factory::group_impl_factory (C++ function), 69
 cui::fcl::groups (C++ type), 68
 cui::fcl::groups::colours_and_fonts (C++ member), 68
 cui::fcl::groups::layout (C++ member), 68
 cui::fcl::groups::title_scripts (C++ member), 68
 cui::fcl::groups::toolbars (C++ member), 68
 cui::fcl::t_export_feedback (C++ class), 69
 cui::fcl::t_export_feedback::add_required_panel (C++ function), 69
 cui::fcl::t_export_feedback::add_required_panels (C++ function), 69
 cui::fcl::t_import_feedback (C++ class), 69
 cui::fcl::t_import_feedback::add_required_panel (C++ function), 69
 cui::fonts (C++ type), 55
 cui::fonts::client::create_by_guid (C++ function), 56
 cui::fonts::client::factory (C++ class), 56
 cui::fonts::client::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 56
 cui::fonts::client::get_client_guid (C++ function), 56
 cui::fonts::client::get_default_font_type (C++ function), 56
 cui::fonts::client::get_name (C++ function), 56
 cui::fonts::client::on_font_changed (C++ function), 56
 cui::fonts::common_callback (C++ class), 56
 cui::fonts::common_callback::on_font_changed (C++ function), 56
 cui::fonts::font_mode_t (C++ enum), 55
 cui::fonts::font_mode_t::font_mode_common_items (C++ enumerator), 55
 cui::fonts::font_mode_t::font_mode_common_labels (C++ enumerator), 55
 cui::fonts::font_mode_t::font_mode_custom (C++ enumerator), 55
 cui::fonts::font_mode_t::font_mode_system (C++ enumerator), 55
 cui::fonts::font_type_flag_t (C++ enum), 55
 cui::fonts::font_type_flag_t::font_type_flag_items (C++ enumerator), 55

cui::fonts::font_type_flag_t::font_type_flag_labels (C++ member), 71
 (C++ enumerator), 55
 cui::fonts::font_type_t (C++ enum), 55
 cui::fonts::font_type_t::font_type_items (C++ enumerator), 55
 cui::fonts::font_type_t::font_type_labels (C++ enumerator), 55
 cui::fonts::helper (C++ class), 56
 cui::fonts::helper::get_font (C++ function), 56
 cui::fonts::helper::helper (C++ function), 56
 cui::fonts::manager (C++ class), 56
 cui::fonts::manager::deregister_common_callback (C++ function), 57
 cui::fonts::manager::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 57
 cui::fonts::manager::get_font (C++ function), 56, 57
 cui::fonts::manager::register_common_callback (C++ function), 56
 cui::fonts::manager::set_font (C++ function), 56
 cui::fonts::manager_v2 (C++ class), 57
 cui::fonts::manager_v2::deregister_common_callback (C++ function), 57
 cui::fonts::manager_v2::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 57
 cui::fonts::manager_v2::get_client_font (C++ function), 57
 cui::fonts::manager_v2::get_client_font_handle (C++ function), 57
 cui::fonts::manager_v2::get_common_font (C++ function), 57
 cui::fonts::manager_v2::get_common_font_handle (C++ function), 57
 cui::fonts::manager_v2::register_common_callback (C++ function), 57
 cui::fonts::manager_v2::set_client_font (C++ function), 57
 cui::global_variable (C++ class), 75
 cui::global_variable::get_name (C++ function), 75
 cui::global_variable::get_value (C++ function), 75
 cui::global_variable::global_variable (C++ function), 75
 cui::global_variable_list (C++ class), 75
 cui::global_variable_list::~global_variable_list (C++ function), 75
 cui::global_variable_list::add_item (C++ function), 75
 cui::global_variable_list::find_by_name (C++ function), 75
 cui::panels (C++ type), 71
 cui::panels::guid_artwork_view (C++ member), 71
 cui::panels::guid_filter (C++ member), 71
 cui::panels::guid_horizontal_splitter (C++ member), 71
 cui::panels::guid_item_details (C++ member), 71
 cui::panels::guid_item_properties (C++ member), 71
 cui::panels::guid_playlist_switcher (C++ member), 71
 cui::panels::guid_playlist_tabs (C++ member), 71
 cui::panels::guid_playlist_view (C++ member), 71
 cui::panels::guid_playlist_view_v2 (C++ member), 71
 cui::panels::guid_vertical_splitter (C++ member), 71
 cui::strings (C++ type), 77
 cui::strings::guid_global_variables (C++ member), 78
 cui::titleformat_hook_global_variables (C++ class), 75
 cui::titleformat_hook_global_variables::process_field (C++ function), 75
 cui::titleformat_hook_global_variables::process_function (C++ function), 75
 cui::titleformat_hook_global_variables::titleformat_hook_global_variables (C++ function), 75
 cui::toolbars (C++ type), 72
 cui::toolbars::guid_buttons (C++ member), 72
 cui::toolbars::guid_filter_search_bar (C++ member), 72
 cui::toolbars::guid_menu (C++ member), 72
 cui::toolbars::guid_playback_order (C++ member), 72
 cui::toolbars::guid_seek_bar (C++ member), 72
 cui::toolbars::guid_spectrum_analyser (C++ member), 72
 cui::toolbars::guid_volume_control (C++ member), 72
 cui::visualisations (C++ type), 72
 cui::visualisations::guid_spectrum_analyser (C++ member), 73

U

uie::button (C++ class), 60
 uie::button::deregister_callback (C++ function), 62
 uie::button::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 62
 uie::button::get_button_state (C++ function), 62
 uie::button::get_button_type (C++ function), 61
 uie::button::get_command_state_count (C++ function), 62

uie::button::get_command_state_index (C++ function), 62
 uie::button::get_command_state_name (C++ function), 62
 uie::button::get_guid_type (C++ function), 60
 uie::button::get_item_bitmap (C++ function), 61
 uie::button::get_item_guid (C++ function), 60
 uie::button::get_menu_items (C++ function), 61
 uie::button::register_callback (C++ function), 62
 uie::button_callback (C++ class), 64
 uie::button_callback::on_button_state_change (C++ function), 65
 uie::button_callback::on_command_state_change (C++ function), 65
 uie::button_factory (C++ class), 65
 uie::button_v2 (C++ class), 62
 uie::button_v2::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 63
 uie::button_v2::get_item_bitmap (C++ function), 63
 uie::button_v2::handle_type_t (C++ enum), 63
 uie::button_v2::handle_type_t::handle_type_bitmap (C++ enumerator), 63
 uie::button_v2::handle_type_t::handle_type_icon (C++ enumerator), 63
 uie::container_uie_window_v3 (C++ type), 34
 uie::container_uie_window_v3_t (C++ class), 32
 uie::container_uie_window_v3_t::create_or_transfer_window (C++ function), 33
 uie::container_uie_window_v3_t::destroy_window (C++ function), 34
 uie::container_uie_window_v3_t::get_host (C++ function), 33
 uie::container_uie_window_v3_t::get_window_config (C++ function), 33
 uie::container_uie_window_v3_t::get_wnd (C++ function), 33
 uie::container_uie_window_v3_t::is_available (C++ function), 33
 uie::container_uie_window_v3_t::on_message (C++ function), 33
 uie::container_window_v3 (C++ class), 32
 uie::container_window_v3::container_window_v3 (C++ function), 32
 uie::container_window_v3::create (C++ function), 32
 uie::container_window_v3::deregister_class (C++ function), 32
 uie::container_window_v3::destroy (C++ function), 32
 uie::container_window_v3::get_wnd (C++ function), 32
 uie::container_window_v3::operator= (C++ function), 32
 uie::container_window_v3_config (C++ struct), 31
 uie::container_window_v3_config::class_background (C++ member), 32
 uie::container_window_v3_config::class_cursor (C++ member), 32
 uie::container_window_v3_config::class_extra_wnd_bytes (C++ member), 32
 uie::container_window_v3_config::class_name (C++ member), 31
 uie::container_window_v3_config::class_styles (C++ member), 32
 uie::container_window_v3_config::container_window_v3_config (C++ function), 31
 uie::container_window_v3_config::extended_window_styles (C++ member), 32
 uie::container_window_v3_config::forward_wm_settingchange (C++ member), 31
 uie::container_window_v3_config::invalidate_children_on_move (C++ member), 31
 uie::container_window_v3_config::use_transparent_background (C++ member), 31
 uie::container_window_v3_config::window_styles (C++ member), 31
 uie::custom_button (C++ class), 64
 uie::custom_button::execute (C++ function), 64
 uie::custom_button::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 64
 uie::custom_button::g_button_get_name (C++ function), 64
 uie::custom_button::get_description (C++ function), 64
 uie::custom_button::get_guid_type (C++ function), 64
 uie::custom_button::get_name (C++ function), 64
 uie::extension_base (C++ class), 13
 uie::extension_base::export_config (C++ function), 14
 uie::extension_base::export_config_to_array (C++ function), 16
 uie::extension_base::get_config (C++ function), 14
 uie::extension_base::get_config_as_array (C++ function), 16
 uie::extension_base::get_config_to_array (C++ function), 15
 uie::extension_base::get_extension_guid (C++ function), 13
 uie::extension_base::get_menu_items (C++ function), 15
 uie::extension_base::get_name (C++ function), 13
 uie::extension_base::have_config_popup (C++ function), 15

uie::extension_base::import_config (C++ function), 14
uie::extension_base::import_config_from_ptr (C++ function), 15
uie::extension_base::set_config (C++ function), 13
uie::extension_base::set_config_from_ptr (C++ function), 15
uie::extension_base::show_config_popup (C++ function), 15
uie::menu_button (C++ class), 63
uie::menu_button::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 64
uie::menu_button::select_subcommand (C++ function), 64
uie::menu_hook_impl (C++ class), 41
uie::menu_hook_impl::add_node (C++ function), 42
uie::menu_hook_impl::execute (C++ function), 42
uie::menu_hook_impl::execute_by_id (C++ function), 42
uie::menu_hook_impl::get_child (C++ function), 42
uie::menu_hook_impl::get_children_count (C++ function), 42
uie::menu_hook_impl::get_description (C++ function), 42
uie::menu_hook_impl::get_display_data (C++ function), 42
uie::menu_hook_impl::get_type (C++ function), 42
uie::menu_hook_impl::win32_build_menu (C++ function), 42
uie::menu_hook_t (C++ class), 41
uie::menu_hook_t::add_node (C++ function), 41
uie::menu_node_command_t (C++ class), 38
uie::menu_node_command_t::get_child (C++ function), 39
uie::menu_node_command_t::get_children_count (C++ function), 39
uie::menu_node_command_t::get_type (C++ function), 39
uie::menu_node_configure (C++ class), 41
uie::menu_node_configure::execute (C++ function), 41
uie::menu_node_configure::get_description (C++ function), 41
uie::menu_node_configure::get_display_data (C++ function), 41
uie::menu_node_configure::menu_node_configure (C++ function), 41
uie::menu_node_popup_t (C++ class), 39
uie::menu_node_popup_t::execute (C++ function), 39
uie::menu_node_popup_t::get_description (C++ function), 39
uie::menu_node_popup_t::get_type (C++ function), 39
uie::menu_node_separator_t (C++ class), 39
uie::menu_node_separator_t::execute (C++ function), 40
uie::menu_node_separator_t::get_child (C++ function), 40
uie::menu_node_separator_t::get_children_count (C++ function), 40
uie::menu_node_separator_t::get_description (C++ function), 40
uie::menu_node_separator_t::get_display_data (C++ function), 40
uie::menu_node_separator_t::get_type (C++ function), 40
uie::menu_node_t (C++ class), 37
uie::menu_node_t::execute (C++ function), 38
uie::menu_node_t::get_child (C++ function), 38
uie::menu_node_t::get_children_count (C++ function), 38
uie::menu_node_t::get_description (C++ function), 38
uie::menu_node_t::get_display_data (C++ function), 38
uie::menu_node_t::get_type (C++ function), 38
uie::menu_node_t::state_t (C++ enum), 37
uie::menu_node_t::state_t::state_checked (C++ enumerator), 37
uie::menu_node_t::state_t::state_disabled (C++ enumerator), 37
uie::menu_node_t::state_t::state_disabled_grey (C++ enumerator), 37
uie::menu_node_t::state_t::state_greyed (C++ enumerator), 37
uie::menu_node_t::state_t::state_radio (C++ enumerator), 38
uie::menu_node_t::state_t::state_radiochecked (C++ enumerator), 38
uie::menu_node_t::type_t (C++ enum), 38
uie::menu_node_t::type_t::type_command (C++ enumerator), 38
uie::menu_node_t::type_t::type_popup (C++ enumerator), 38
uie::menu_node_t::type_t::type_separator (C++ enumerator), 38
uie::menu_window (C++ class), 20
uie::menu_window::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 21
uie::menu_window::hide_accelerators (C++ function), 21
uie::menu_window::is_menu_focused (C++ function), 20
uie::menu_window::on_menuchar (C++ function), 20
uie::menu_window::set_focus (C++ function), 20

uie::menu_window::show_accelerators (C++ function), 28
 function), 20
 uie::menu_window_v2 (C++ class), 21
 uie::menu_window_v2::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 21
 uie::menu_window_v2::get_previous_focus_window (C++ function), 21
 uie::playlist_window (C++ class), 20
 uie::playlist_window::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 20
 uie::playlist_window::set_focus (C++ function), 20
 uie::simple_command_menu_node (C++ class), 40
 uie::simple_command_menu_node::execute (C++ function), 41
 uie::simple_command_menu_node::get_description (C++ function), 40
 uie::simple_command_menu_node::get_display_data (C++ function), 40
 uie::simple_command_menu_node::simple_command_menu_node (C++ function), 40
 uie::splitter_item_full_t (C++ class), 27
 uie::splitter_item_full_t::get_class_guid (C++ function), 27
 uie::splitter_item_full_t::get_title (C++ function), 27
 uie::splitter_item_full_t::m_autohide (C++ member), 27
 uie::splitter_item_full_t::m_caption_orientation (C++ member), 27
 uie::splitter_item_full_t::m_custom_title (C++ member), 27
 uie::splitter_item_full_t::m_hidden (C++ member), 27
 uie::splitter_item_full_t::m_locked (C++ member), 27
 uie::splitter_item_full_t::m_show_caption (C++ member), 27
 uie::splitter_item_full_t::m_show_toggle_area (C++ member), 27
 uie::splitter_item_full_t::m_size (C++ member), 27
 uie::splitter_item_full_t::query (C++ function), 27
 uie::splitter_item_full_t::set_title (C++ function), 27
 uie::splitter_item_full_v2_t (C++ class), 27
 uie::splitter_item_full_v2_t::get_class_guid (C++ function), 28
 uie::splitter_item_full_v2_t::m_size_v2 (C++ member), 28
 uie::splitter_item_full_v2_t::m_size_v2_dpi (C++ member), 28
 uie::splitter_item_full_v2_t::query (C++

function), 28
 uie::splitter_item_full_v3_impl_t (C++ class), 29
 uie::splitter_item_full_v3_impl_t::get_extra_data (C++ function), 29
 uie::splitter_item_full_v3_impl_t::get_extra_data_format_id (C++ function), 29
 uie::splitter_item_full_v3_impl_t::m_extra_data (C++ member), 29
 uie::splitter_item_full_v3_impl_t::m_extra_data_format_id (C++ member), 29
 uie::splitter_item_full_v3_t (C++ class), 28
 uie::splitter_item_full_v3_t::get_class_guid (C++ function), 29
 uie::splitter_item_full_v3_t::get_extra_data (C++ function), 28
 uie::splitter_item_full_v3_t::get_extra_data_format_id (C++ function), 28
 uie::splitter_item_full_v3_t::query (C++ function), 28
 uie::splitter_item_simple (C++ class), 26
 uie::splitter_item_simple::get_panel_config (C++ function), 26
 uie::splitter_item_simple::get_panel_guid (C++ function), 26
 uie::splitter_item_simple::get_window_ptr (C++ function), 26
 uie::splitter_item_simple::set_panel_config (C++ function), 26
 uie::splitter_item_simple::set_panel_guid (C++ function), 26
 uie::splitter_item_simple::set_window_ptr (C++ function), 26
 uie::splitter_item_t (C++ class), 26
 uie::splitter_item_t::~splitter_item_t (C++ function), 26
 uie::splitter_item_t::get_panel_config (C++ function), 26
 uie::splitter_item_t::get_panel_config_to_array (C++ function), 26
 uie::splitter_item_t::get_panel_guid (C++ function), 26
 uie::splitter_item_t::get_window_ptr (C++ function), 26
 uie::splitter_item_t::query (C++ function), 26
 uie::splitter_item_t::set_panel_config (C++ function), 26
 uie::splitter_item_t::set_panel_config_from_ptr (C++ function), 26
 uie::splitter_item_t::set_panel_guid (C++ function), 26
 uie::splitter_window (C++ class), 23
 uie::splitter_window::add_panel (C++ function), 24

uie::splitter_window::bool_autohide (C++ member), 24
 uie::splitter_window::bool_hidden (C++ member), 24
 uie::splitter_window::bool_locked (C++ member), 24
 uie::splitter_window::bool_show_caption (C++ member), 24
 uie::splitter_window::bool_show_toggle_area (C++ member), 24
 uie::splitter_window::bool_use_custom_title (C++ member), 25
 uie::splitter_window::deregister_callback (C++ function), 24
 uie::splitter_window::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 24
 uie::splitter_window::find_by_ptr (C++ function), 24
 uie::splitter_window::get_config_item (C++ function), 23
 uie::splitter_window::get_config_item_supported (C++ function), 23
 uie::splitter_window::get_maximum_panel_count (C++ function), 24
 uie::splitter_window::get_panel (C++ function), 24
 uie::splitter_window::get_panel_count (C++ function), 24
 uie::splitter_window::insert_panel (C++ function), 23
 uie::splitter_window::move_down (C++ function), 24
 uie::splitter_window::move_up (C++ function), 24
 uie::splitter_window::register_callback (C++ function), 24
 uie::splitter_window::remove_panel (C++ function), 23, 24
 uie::splitter_window::replace_panel (C++ function), 24
 uie::splitter_window::set_config_item (C++ function), 23, 24
 uie::splitter_window::set_config_item_t (C++ function), 23
 uie::splitter_window::size_and_dpi (C++ member), 25
 uie::splitter_window::string_custom_title (C++ member), 25
 uie::splitter_window::swap_items (C++ function), 24
 uie::splitter_window::uint32_orientation (C++ member), 24
 uie::splitter_window::uint32_size (C++ member), 25
 uie::splitter_window_v2 (C++ class), 25
 uie::splitter_window_v2::FB2K_MAKE_SERVICE_INTERFACE (C++ function), 25
 uie::splitter_window_v2::get_supported_panels (C++ function), 25
 uie::splitter_window_v2::is_point_ours (C++ function), 25
 uie::t_button_guid (C++ enum), 59
 uie::t_button_guid::BUTTON_GUID_BUTTON (C++ enumerator), 59
 uie::t_button_guid::BUTTON_GUID_MENU_ITEM_CONTEXT (C++ enumerator), 59
 uie::t_button_guid::BUTTON_GUID_MENU_ITEM_MAIN (C++ enumerator), 59
 uie::t_button_state (C++ enum), 59
 uie::t_button_state::BUTTON_STATE_DEFAULT (C++ enumerator), 60
 uie::t_button_state::BUTTON_STATE_ENABLED (C++ enumerator), 60
 uie::t_button_state::BUTTON_STATE_PRESSED (C++ enumerator), 60
 uie::t_button_state::BUTTON_STATE_SHOW_TOOLTIP (C++ enumerator), 60
 uie::t_button_type (C++ enum), 59
 uie::t_button_type::BUTTON_TYPE_DROPDOWN (C++ enumerator), 59
 uie::t_button_type::BUTTON_TYPE_DROPDOWN_ARROW (C++ enumerator), 59
 uie::t_button_type::BUTTON_TYPE_NORMAL (C++ enumerator), 59
 uie::t_mask (C++ enum), 60
 uie::t_mask::MASK_BITMAP (C++ enumerator), 60
 uie::t_mask::MASK_COLOUR (C++ enumerator), 60
 uie::t_mask::MASK_NONE (C++ enumerator), 60
 uie::visualisation (C++ class), 35
 uie::visualisation::create_by_guid (C++ function), 35
 uie::visualisation::disable (C++ function), 35
 uie::visualisation::enable (C++ function), 35
 uie::visualisation::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 35
 uie::visualisation::paint_background (C++ function), 35
 uie::visualisation_factory (C++ class), 36
 uie::visualisation_host (C++ class), 36
 uie::visualisation_host::createPainter (C++ function), 36
 uie::visualisation_host::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 36
 uie::visualisation_host::painter_ptr (C++ type), 36
 uie::visualisation_host::painter_t (C++ class), 36
 uie::visualisation_host::painter_t::get_area (C++ function), 36

uie::visualisation_host::painter_t::get_device_content_text (C++ function), 36
 uie::win32::paint_background_using_parent (C++ function), 34
 uie::window (C++ class), 17
 uie::window::create_by_guid (C++ function), 19
 uie::window::create_or_transfer_window (C++ function), 18
 uie::window::destroy_window (C++ function), 19
 uie::window::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 19
 uie::window::g_on_tab (C++ function), 19
 uie::window::g_process_keydown_keyboard_shortcuts (C++ function), 19
 uie::window::get_category (C++ function), 17
 uie::window::get_description (C++ function), 17
 uie::window::get_is_single_instance (C++ function), 17
 uie::window::get_prefer_multiple_instances (C++ function), 18
 uie::window::get_short_name (C++ function), 17
 uie::window::get_size_limits (C++ function), 19
 uie::window::get_type (C++ function), 17
 uie::window::get_wnd (C++ function), 19
 uie::window::is_available (C++ function), 18
 uie::window_factory (C++ class), 22
 uie::window_factory::~~window_factory (C++ function), 22
 uie::window_factory::instance_create (C++ function), 22
 uie::window_factory::window_factory (C++ function), 22
 uie::window_host (C++ class), 43
 uie::window_host::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 46
 uie::window_host::get_host_guid (C++ function), 43
 uie::window_host::get_keyboard_shortcuts_enabled (C++ function), 44
 uie::window_host::is_resize_supported (C++ function), 44
 uie::window_host::is_visibility_modifiable (C++ function), 45
 uie::window_host::is_visible (C++ function), 45
 uie::window_host::on_size_limit_change (C++ function), 43
 uie::window_host::override_status_text_create (C++ function), 44
 uie::window_host::relinquish_ownership (C++ function), 45
 uie::window_host::request_resize (C++ function), 44
 uie::window_host::set_window_visibility (C++ function), 45
 uie::window_host_factory (C++ class), 47
 uie::window_host_factory_single (C++ class), 47
 uie::window_host_factory_single::operator uie::window_host_ptr (C++ function), 48
 uie::window_host_factory_transparent_single (C++ class), 48
 uie::window_host_with_control (C++ class), 46
 uie::window_host_with_control::FB2K_MAKE_SERVICE_INTERFACE_ENTRYPOINT (C++ function), 47
 uie::window_host_with_control::get_name (C++ function), 47
 uie::window_host_with_control::get_supported_types (C++ function), 46
 uie::window_host_with_control::insert_extension (C++ function), 46
 uie::window_host_with_control::is_available (C++ function), 47