Contents

Microsoft UI Automation

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Microsoft UI Automation is an accessibility framework for Microsoft Windows. It addresses the needs of assistive technology products and automated test frameworks by providing programmatic access to information about the user interface (UI). In addition, UI Automation enables control and application developers to make their products accessible.

This documentation describes the UI Automation API for managed code. For information on programming for UI Automation in C++, see UI Automation for Win32 Applications.

In this section

- Accessibility Best Practices
- UI Automation Fundamentals
- UI Automation Providers for Managed Code
- UI Automation Clients for Managed Code
- UI Automation Control Patterns
- UI Automation Text Pattern
- UI Automation Control Types

Related sections

- Accessibility Samples

Accessibility Best Practices

3/12/2020 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Implementing the following best practices in controls or applications will improve their accessibility for people who use assistive technology devices. Many of these best practices focus on good user interface (UI) design. Each best practice includes implementation information for Windows Presentation Foundation (WPF) controls or applications. In many cases, the work to meet these best practices is already included in WPF controls.

Programmatic Access

Programmatic access involves ensuring that all UI elements are labeled, property values are exposed, and appropriate events are raised. For standard WPF controls, most of this work is already done through AutomationPeer. Custom controls require additional work to ensure that programmatic access is correctly implemented.

Enable Programmatic Access to all UI Elements and Text

User interface (UI) elements should enable programmatic access. If the UI is a standard WPF control, support for programmatic access is included in the control. If the control is a custom control – a control that has been subclassed from a common control or a control that has been subclassed from Control – then you must check the AutomationPeer implementation for areas that may need modification.

Following this best practice allows assistive technology vendors to identify and manipulate elements of your product's UI.

Place Names, Titles, and Descriptions on UI Objects, Frames, and Pages

Assistive technologies, especially screen readers, use the title to understand the location of the frame, object, or page in the navigation scheme. Therefore, the title must be very descriptive. For example, a Web page title of "Microsoft Web Page" is useless if the user has navigated deeply into some particular area. A descriptive title is critical for users who are blind and depend on screen readers. Similarly, for Windows Presentation Foundation (WPF) controls, NameProperty and HelpTextProperty are important for assistive technology devices.

Following this best practice allows assistive technologys to identify and manipulate UI in sample controls and applications.

Ensure Programmatic Events Are Triggered by All UI Activities

Following this best practice allows assistive technologys to listen for changes in the UI and notify the user about these changes.

User Settings

The best practice in this section ensures that controls or applications do not override user settings.

Respect All System-Wide Settings and Do Not Interfere with Accessibility Functions

Users can use the Control Panel to set some system-wide flags; other flags can be set programmatically. These settings should not be changed by controls or applications. Also, applications must support the accessibility settings of their host operating system.

Following this best practice allows users to set accessibility settings and know that those settings will not be changed by applications.

Visual UI Design

Best Practices in this section ensure that controls or applications use color and images effectively and are able to be used by Assistive technologies.

Don't Hard-Code Colors

People who are color blind, have low vision, or are using a black and white screen might not be able to use applications with hard-coded colors.

Following this best practice allows users to adjust color combinations based on individual needs.

Support High Contrast and all System Display Attributes

Applications should not disrupt or disable user-selected, system-wide contrast settings, color selections, or other system-wide display settings and attributes. System-wide settings adopted by a user enhance the accessibility of applications, so they should not be disabled or disregarded by applications. Color should be used in their correct foreground-on-background combination to provide proper contrast. Unrelated colors should not be mixed, and colors should not be reversed.

Many users require specific high-contrast combinations, such as white text on a black background. Drawing these reversed, as black text on a white background causes the background to bleed over the foreground and can make reading difficult for some users.

Ensure All UI Correctly Scales by Any DPI Setting

Ensure that all UI can correctly scale by any dots per inch (dpi) setting. Also, ensure that UI elements fit in a screen of 1024 x 768 with 120 dots per inch (dpi).

Navigation

Best Practices in this section ensure that navigation has been addressed for controls and applications.

Provide Keyboard Interface for All UI Elements

Tab stops, especially when carefully planned, give users another way to navigate the UI.

Applications should provide the following keyboard interfaces:

- tab stops for all controls that the user can interact with, such as buttons, links, or list boxes
- logical tab order

Show the Keyboard Focus

Users need to know which object has the keyboard focus so that they can anticipate the effect of their keystrokes. To highlight the keyboard focus, use colors, fonts, or graphics such as rectangles or magnification. To audibly highlight the keyboard focus, change the volume, pitch or tonal quality.

To avoid confusion, applications should hide all visual focus indicators and dim selections that are located in inactive windows (or panes).

Applications should do the following with keyboard focus:

- one item should always have keyboard focus
- keyboard focus should be visible and obvious

- selections and/or focused items should be visually highlighted

Support Navigation Standards and Powerful Navigation Schemes

Different aspects of keyboard navigation provide different ways for users to navigate the UI.

Applications should provide the following keyboard interfaces:

- shortcut keys and underlined access keys for all commands, menus and controls
- keyboard shortcuts to important links
- all menu items have an access key; all buttons have accelerator keys, all commands have an accelerator key.

Do Not Let Mouse Location Interfere with Keyboard Navigation

Mouse location should not interfere with keyboard navigation. For example, if the mouse is positioned someplace and the user is navigating with the keyboard, a mouse click should not happen unless initiated by the user.

Multimodal Interface

Best Practices in this section ensure that application UI includes alternatives for visual elements.

Provide User-Selectable Equivalents for Non-Text Elements

For each non-text element, provide a user-selectable equivalent for text, transcripts, or audio descriptions, such as alt text, captions, or visual feedback.

Non-text elements cover a wide range of UI elements including: images, image map regions, animations, applets, frames, scripts, graphical buttons, sounds, stand-alone audio files and video. Non-text elements are important when they contain visual information, speech, or general audio information that the user needs access to in order to understand the content of the UI.

Use Color but Also Provide Alternatives to Color

Use color to enhance, emphasize, or reiterate information shown by other means, but do not communicate information by using color alone. Users who are color blind or have a monochrome display need alternatives to color.

Use Standard Input APIs with Device-Independent Calls

Device-independent calls ensure keyboard and mouse feature equality, while providing assistive technology with needed information about the UI.

See also

- System.Windows.Automation.Peers
- NumericUpDown Custom Control with Theme and UI Automation Support Sample
- Guidelines for Keyboard User Interface Design

UI Automation Fundamentals

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section contains high-level overviews of the Microsoft UI Automation API.

In This Section

Reference

System.Windows.Automation

System.Windows.Automation.Provider

System.Windows.Automation.Text

UIAutomationClientsideProviders

UI Automation Overview

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Microsoft UI Automation is the new accessibility framework for Microsoft Windows, available on all operating systems that support Windows Presentation Foundation (WPF).

UI Automation provides programmatic access to most user interface (UI) elements on the desktop, enabling assistive technology products such as screen readers to provide information about the UI to end users and to manipulate the UI by means other than standard input. UI Automation also allows automated test scripts to interact with the UI.

NOTE

UI Automation does not enable communication between processes started by different users through the **Run as** command.

UI Automation client applications can be written with the assurance that they will work on multiple frameworks. The UI Automation core masks any differences in the frameworks that underlie various pieces of UI. For example, the `Content` property of a WPF button, the `Caption` property of a Win32 button, and the `ALT` property of an HTML image are all mapped to a single property, Name, in the UI Automation view.

UI Automation provides full functionality on supported Windows operating systems running the .NET Framework (see .NET Framework system requirements or versions of .NET Core starting with .NET Core 3.0.

UI Automation providers offer some support for Microsoft Active Accessibility client applications through a built-in bridging service.

Providers and Clients

UI Automation has four main components, as shown in the following table.

COMPONENT	DESCRIPTION
Provider API (UIAutomationProvider.dll and UIAutomationTypes.dll)	A set of interface definitions that are implemented by UI Automation providers, objects that provide information about UI elements and respond to programmatic input.
Client API (UIAutomationClient.dll and UIAutomationTypes.dll)	A set of types for managed code that enables UI Automation client applications to obtain information about the UI and to send input to controls.
UiAutomationCore.dll	The underlying code (sometimes called the UI Automation core) that handles communication between providers and clients.

COMPONENT	DESCRIPTION
UIAutomationClientsideProviders.dll	A set of UI Automation providers for standard legacy controls. (WPF controls have native support for UI Automation.) This support is automatically available to client applications.

From the software developer's perspective, there are two ways of using UI Automation: to create support for custom controls (using the provider API), and creating applications that use the UI Automation core to communicate with UI elements (using the client API). Depending on your focus, you should refer to different parts of the documentation. You can learn more about the concepts and gain practical how-to knowledge in the following sections.

SECTION	SUBJECT MATTER	AUDIENCE
UI Automation Fundamentals (this section)	Broad overviews of the concepts.	All.
UI Automation Providers for Managed Code	Overviews and how-to topics to help you use the provider API.	Control developers.
UI Automation Clients for Managed Code	Overviews and how-to topics to help you use the client API.	Client application developers.
UI Automation Control Patterns	Information about how control patterns should be implemented by providers, and what functionality is available to clients.	All.
UI Automation Text Pattern	Information about how the Text control pattern should be implemented by providers, and what functionality is available to clients.	All.
UI Automation Control Types	Information about the properties and control patterns supported by different control types.	All.

The following table lists UI Automation namespaces, the DLLs that contain them, and the audience that uses them.

NAMESPACE	REFERENCED DLLS	AUDIENCE
System.Windows.Automation	UIAutomationClientUIAutomationTypes	UI Automation client developers; used to find AutomationElement objects, register for UI Automation events, and work with UI Automation control patterns.
System.Windows.Automation.Provider	UIAutomationProviderUIAutomationTypes	Developers of UI Automation providers for frameworks other than WPF.

NAMESPACE	REFERENCED DLLS	AUDIENCE
System.Windows.Automation.Text	UIAutomationClientUIAutomationTypes	Developers of UI Automation providers for frameworks other than WPF; used to implement the TextPattern control pattern.
System.Windows.Automation.Peers	PresentationFramework	Developers of UI Automation providers for WPF.

UI Automation Model

UI Automation exposes every piece of the UI to client applications as an AutomationElement. Elements are contained in a tree structure, with the desktop as the root element. Clients can filter the raw view of the tree as a control view or a content view. Applications can also create custom views.

AutomationElement objects expose common properties of the UI elements they represent. One of these properties is the control type, which defines its basic appearance and functionality as a single recognizable entity: for example, a button or check box.

In addition, elements expose control patterns that provide properties specific to their control types. Control patterns also expose methods that enable clients to get further information about the element and to provide input.

NOTE

There is not a one-to-one correspondence between control types and control patterns. A control pattern may be supported by multiple control types, and a control may support multiple control patterns, each of which exposes different aspects of its behavior. For example, a combo box has at least two control patterns: one that represents its ability to expand and collapse, and another that represents the selection mechanism. For specifics, see UI Automation Control Types.

UI Automation also provides information to client applications through events. Unlike WinEvents, UI Automation events are not based on a broadcast mechanism. UI Automation clients register for specific event notifications and can request that specific UI Automation properties and control pattern information be passed into their event handlers. In addition, a UI Automation event contains a reference to the element that raised it. Providers can improve performance by raising events selectively, depending on whether any clients are listening.

See also

- UI Automation Tree Overview
- UI Automation Control Patterns Overview
- UI Automation Properties Overview
- UI Automation Events Overview
- UI Automation Security Overview

UI Automation and Microsoft Active Accessibility

3/12/2020 • 10 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Microsoft Active Accessibility was the earlier solution for making applications accessible. Microsoft UI Automation is the new accessibility model for Microsoft Windows and is intended to address the needs of assistive technology products and automated testing tools. UI Automation offers many improvements over Active Accessibility.

This topic includes the main features of UI Automation and explains how these features differ from Active Accessibility.

Programming Languages

<Active Accessibility is based on the Component Object Model (COM) with support for dual interfaces, and is therefore programmable in C/C++, Microsoft Visual Basic 6.0, and scripting languages. UI Automation (including the client-side provider library for standard controls) is written in managed code, and UI Automation client applications are most easily programmed using C# or Visual Basic .NET. UI Automation providers, which are interface implementations, can be written in managed code or in C/C++.

Support in Windows Presentation Foundation

Windows Presentation Foundation (WPF) is the new model for creating user interfaces. WPF elements do not contain native support for Active Accessibility; however, they do support UI Automation, which includes bridging support for Active Accessibility clients. Only clients written specifically for UI Automation can take full advantage of the accessibility features of WPF, such as the rich support for text.

Servers and Clients

In Active Accessibility, servers and clients communicate directly, largely through the server's implementation of `IAccessible`.

In UI Automation, a core service lies between the server (called a provider) and the client. The core service makes calls to the interfaces implemented by providers and provides additional services such as generating unique runtime identifiers for elements. Client applications use library functions to call the UI Automation service.

UI Automation providers can provide information to Active Accessibility clients, and Active Accessibility servers can provide information to UI Automation client applications. However, because Active Accessibility does not expose as much information as UI Automation, the two models are not fully compatible.

UI Elements

Active Accessibility presents UI elements either as an `IAccessible` interface or as a child identifier. It is difficult to compare two `IAccessible` pointers to determine if they refer to the same element.

In UI Automation, every element is represented as an AutomationElement object. Comparison is done by using the equality operator or the Equals method, both of which compare the unique runtime identifiers of the elements.

Tree Views and Navigation

The user interface (UI) elements on the screen can be seen as a tree structure with the desktop as the root, application windows as immediate children, and elements within applications as further descendants.

In Active Accessibility, many automation elements that are irrelevant to end users are exposed in the tree. Client applications have to look at all the elements to determine which are meaningful.

UI Automation client applications see the UI through a filtered view. The view contains only elements of interest: those that give information to the user or enable interaction. Predefined views of only control elements and only content elements are available; in addition, applications can define custom views. UI Automation simplifies the task of describing the UI to the user and helping the user interact with the application.

Navigation between elements, in Active Accessibility, is either spatial (for example, moving to the element that lies to the left on the screen), logical (for example, moving to the next menu item, or the next item in the tab order within a dialog box), or hierarchical (for example, moving the first child in a container, or from the child to its parent). Hierarchical navigation is complicated by the fact that child elements are not always objects that implement `IAccessible`.

In UI Automation, all UI elements are AutomationElement objects that support the same basic functionality. (From the standpoint of the provider, they are objects that implement an interface inherited from IRawElementProviderSimple.) Navigation is mainly hierarchical: from parents to children, and from one sibling to the next. (Navigation between siblings has a logical element, as it may follow the tab order.) You can navigate from any starting-point, using any filtered view of the tree, by using the TreeWalker class. You can also navigate to particular children or descendants by using FindFirst and FindAll; for example, it is very easy to retrieve all elements within a dialog box that support a specified control pattern.

Navigation in UI Automation is more consistent than in Active Accessibility. Some elements such as drop-down lists and pop-up windows appear twice in the Active Accessibility tree, and navigation from them may have unexpected results. It is actually impossible to properly implement Active Accessibility for a rebar control. UI Automation enables reparenting and repositioning, so that an element can be placed anywhere in the tree despite the hierarchy imposed by ownership of windows.

Roles and Control Types

Active Accessibility uses the `accRole` property (`IAccessible::get_actRole`) to retrieve a description of the element's role in the UI, such as ROLE_SYSTEM_SLIDER or ROLE_SYSTEM_MENUITEM. The role of an element is the main clue to its available functionality. Interaction with a control is achieved by using fixed methods such as `IAccessible::accSelect` and `IAccessible::accDoDefaultAction`. The interaction between the client application and the UI is limited to what can be done through `IAccessible`.

In contrast, UI Automation largely decouples the control type of the element (described by the ControlType property) from its expected functionality. Functionality is determined by the control patterns that are supported by the provider through its implementation of specialized interfaces. Control patterns can be combined to describe the full set of functionality supported by a particular UI element. Some providers are required to support a particular control pattern; for example, the provider for a check box must support the Toggle control pattern. Other providers are required to support one or more of a set of control patterns; for example, a button must support either Toggle or Invoke. Still others support no control patterns at all; for example, a pane that cannot be moved, resized, or docked does not have any control patterns.

UI Automation supports custom controls, which are identified by the Custom property and can be described by the LocalizedControlTypeProperty property.

The following table shows the mapping of Active Accessibility roles to UI Automation control types.

ACTIVE ACCESSIBILITY ROLE	UI AUTOMATION CONTROL TYPE
ROLE_SYSTEM_PUSHBUTTON	Button
ROLE_SYSTEM_CLIENT	Calendar
ROLE_SYSTEM_CHECKBUTTON	Check box
ROLE_SYSTEM_COMBOBOX	Combo box
ROLE_SYSTEM_CLIENT	Custom
ROLE_SYSTEM_LIST	Data grid
ROLE_SYSTEM_LISTITEM	Data item
ROLE_SYSTEM_DOCUMENT	Document
ROLE_SYSTEM_TEXT	Edit
ROLE_SYSTEM_GROUPING	Group
ROLE_SYSTEM_LIST	Header
ROLE_SYSTEM_COLUMNHEADER	Header item
ROLE_SYSTEM_LINK	Hyperlink
ROLE_SYSTEM_GRAPHIC	Image
ROLE_SYSTEM_LIST	List
ROLE_SYSTEM_LISTITEM	List item
ROLE_SYSTEM_MENUPOPUP	Menu
ROLE_SYSTEM_MENUBAR	Menu bar
ROLE_SYSTEM_MENUITEM	Menu item
ROLE_SYSTEM_PANE	Pane
ROLE_SYSTEM_PROGRESSBAR	Progress bar
ROLE_SYSTEM_RADIOBUTTON	Radio button
ROLE_SYSTEM_SCROLLBAR	Scroll bar
ROLE_SYSTEM_SEPARATOR	Separator
ROLE_SYSTEM_SLIDER	Slider

ACTIVE ACCESSIBILITY ROLE	UI AUTOMATION CONTROL TYPE
ROLE_SYSTEM_SPINBUTTON	Spinner
ROLE_SYSTEM_SPLITBUTTON	Split button
ROLE_SYSTEM_STATUSBAR	Status bar
ROLE_SYSTEM_PAGETABLIST	Tab
ROLE_SYSTEM_PAGETAB	Tab item
ROLE_SYSTEM_TABLE	Table
ROLE_SYSTEM_STATICTEXT	Text
ROLE_SYSTEM_INDICATOR	Thumb
ROLE_SYSTEM_TITLEBAR	Title bar
ROLE_SYSTEM_TOOLBAR	Tool bar
ROLE_SYSTEM_TOOLTIP	ToolTip
ROLE_SYSTEM_OUTLINE	Tree
ROLE_SYSTEM_OUTLINEITEM	Tree item
ROLE_SYSTEM_WINDOW	Window

For more information about the different control types, see UI Automation Control Types.

States and Properties

In Active Accessibility, elements support a common set of properties, and some properties (such as `accState`) must describe very different things, depending on the element's role. Servers must implement all methods of `IAccessible` that return a property, even those that are not relevant to the element.

UI Automation defines many more properties, some of which correspond to states in Active Accessibility. Some are common to all elements, but others are specific to control types and control patterns. Properties are distinguished by unique identifiers, and most properties can be retrieved by using a single method, GetCurrentPropertyValue or GetCachedPropertyValue. Many properties are also easily retrievable from the Current and Cached property accessors.

A UI Automation provider does not have to implement irrelevant properties, but can simply return a `null` value for any properties it does not support. Also, the UI Automation core service can obtain some properties from the default window provider, and these are amalgamated with properties explicitly implemented by the provider.

As well as supporting many more properties, UI Automation supplies better performance by allowing multiple properties to be retrieved with a single cross-process call.

The following table shows the correspondence between properties in the two models.

ACTIVE ACCESSIBILITY PROPERTY ACCESSOR	UI AUTOMATION PROPERTY ID	REMARKS
`get_accKeyboardShortcut`	AccessKeyProperty or AcceleratorKeyProperty	`AccessKeyProperty` takes precedence if both are present.
`get_accName`	NameProperty	
`get_accRole`	ControlTypeProperty	See the previous table for mapping of roles to control types.
`get_accValue`	ValuePattern.ValueProperty RangeValuePattern.ValueProperty	Valid only for control types that support ValuePattern or RangeValuePattern. RangeValue values are normalized to 0-100, to be consistent with MSAA behavior. Value items use a string.
`get_accHelp`	HelpTextProperty	
`accLocation`	BoundingRectangleProperty	
`get_accDescription`	Not supported in UI Automation	`accDescription` did not have a clear specification within MSAA, which resulted in providers placing different pieces of information in this property.
`get_accHelpTopic`	Not supported in UI Automation	

The following table shows which UI Automation properties correspond to Active Accessibility state constants.

ACTIVE ACCESSIBILITY STATE	UI AUTOMATION PROPERTY	TRIGGERS STATE CHANGE?
STATE_SYSTEM_CHECKED	For check box, ToggleStateProperty For radio button, IsSelectedProperty	Y
STATE_SYSTEM_COLLAPSED	ExpandCollapseState = Collapsed	Y
STATE_SYSTEM_EXPANDED	ExpandCollapseState = Expanded or PartiallyExpanded	Y
STATE_SYSTEM_FOCUSABLE	IsKeyboardFocusableProperty	N
STATE_SYSTEM_FOCUSED	HasKeyboardFocusProperty	N
STATE_SYSTEM_HASPOPUP	ExpandCollapsePattern for menu items	N
STATE_SYSTEM_INVISIBLE	IsOffscreenProperty = True and GetClickablePoint causes NoClickablePointException	N
STATE_SYSTEM_LINKED	ControlTypeProperty = Hyperlink	N

ACTIVE ACCESSIBILITY STATE	UI AUTOMATION PROPERTY	TRIGGERS STATE CHANGE?
STATE_SYSTEM_MIXED	ToggleState = Indeterminate	N
STATE_SYSTEM_MOVEABLE	CanMoveProperty	N
STATE_SYSTEM_MUTLISELECTABLE	CanSelectMultipleProperty	N
STATE_SYSTEM_OFFSCREEN	IsOffscreenProperty = True	N
STATE_SYSTEM_PROTECTED	IsPasswordProperty	N
STATE_SYSTEM_READONLY	RangeValuePattern.IsReadOnlyProperty and ValuePattern.IsReadOnlyProperty	N
STATE_SYSTEM_SELECTABLE	SelectionItemPattern is supported	N
STATE_SYSTEM_SELECTED	IsSelectedProperty	N
STATE_SYSTEM_SIZEABLE	CanResize	N
STATE_SYSTEM_UNAVAILABLE	IsEnabledProperty	Y

The following states either were not implemented by most Active Accessibility control servers or have no equivalent in UI Automation.

ACTIVE ACCESSIBILITY STATE	REMARKS
STATE_SYSTEM_BUSY	Not available in UI Automation
STATE_SYSTEM_DEFAULT	Not available in UI Automation
STATE_SYSTEM_ANIMATED	Not available in UI Automation
STATE_SYSTEM_EXTSELECTABLE	Not widely implemented by Active Accessibility servers
STATE_SYSTEM_MARQUEED	Not widely implemented by Active Accessibility servers
STATE_SYSTEM_SELFVOICING	Not widely implemented by Active Accessibility servers
STATE_SYSTEM_TRAVERSED	Not available in UI Automation
STATE_SYSTEM_ALERT_HIGH	Not widely implemented by Active Accessibility servers
STATE_SYSTEM_ALERT_MEDIUM	Not widely implemented by Active Accessibility servers
STATE_SYSTEM_ALERT_LOW	Not widely implemented by Active Accessibility servers
STATE_SYSTEM_FLOATING	Not widely implemented by Active Accessibility servers
STATE_SYSTEM_HOTTRACKED	Not available in UI Automation

ACTIVE ACCESSIBILITY STATE	REMARKS
STATE_SYSTEM_PRESSED	Not available in UI Automation

For a complete list of UI Automation property identifiers, see UI Automation Properties Overview.

Events

The event mechanism in UI Automation, unlike that in Active Accessibility, does not rely on Windows event routing (which is closely tied in with window handles) and does not require the client application to set up hooks. Subscriptions to events can be fine-tuned not just to particular events but to particular parts of the tree. Providers can also fine-tune their raising of events by keeping track of what events are being listened for.

It is also easier for clients to retrieve the elements that raise events, as these are passed directly to the event callback. Properties of the element are automatically prefetched if a cache request was active when the client subscribed to the event.

The following table shows the correspondence of Active Accessibility WinEvents and UI Automation events.

WINEVENT	UI AUTOMATION EVENT IDENTIFIER
EVENT_OBJECT_ACCELERATORCHANGE	AcceleratorKeyProperty property change
EVENT_OBJECT_CONTENTSCROLLED	VerticalScrollPercentProperty or HorizontalScrollPercentProperty property change on the associated scroll bars
EVENT_OBJECT_CREATE	StructureChangedEvent
EVENT_OBJECT_DEFACTIONCHANGE	No equivalent
EVENT_OBJECT_DESCRIPTIONCHANGE	No exact equivalent; perhaps HelpTextProperty or LocalizedControlTypeProperty property change
EVENT_OBJECT_DESTROY	StructureChangedEvent
EVENT_OBJECT_FOCUS	AutomationFocusChangedEvent
EVENT_OBJECT_HELPCHANGE	HelpTextProperty change
EVENT_OBJECT_HIDE	StructureChangedEvent
EVENT_OBJECT_LOCATIONCHANGE	BoundingRectangleProperty property change
EVENT_OBJECT_NAMECHANGE	NameProperty property change
EVENT_OBJECT_PARENTCHANGE	StructureChangedEvent
EVENT_OBJECT_REORDER	Not consistently used in Active Accessibility. No directly corresponding event is defined in UI Automation.
EVENT_OBJECT_SELECTION	ElementSelectedEvent
EVENT_OBJECT_SELECTIONADD	ElementAddedToSelectionEvent

WINEVENT	UI AUTOMATION EVENT IDENTIFIER
EVENT_OBJECT_SELECTIONREMOVE	ElementRemovedFromSelectionEvent
EVENT_OBJECT_SELECTIONWITHIN	No equivalent
EVENT_OBJECT_SHOW	StructureChangedEvent
EVENT_OBJECT_STATECHANGE	Various property-changed events
EVENT_OBJECT_VALUECHANGE	RangeValuePattern.ValueProperty and ValuePattern.ValueProperty changed
EVENT_SYSTEM_ALERT	No equivalent
EVENT_SYSTEM_CAPTUREEND	No equivalent
EVENT_SYSTEM_CAPTURESTART	No equivalent
EVENT_SYSTEM_CONTEXTHELPEND	No equivalent
EVENT_SYSTEM_CONTEXTHELPSTART	No equivalent
EVENT_SYSTEM_DIALOGEND	WindowClosedEvent
EVENT_SYSTEM_DIALOGSTART	WindowOpenedEvent
EVENT_SYSTEM_DRAGDROPEND	No equivalent
EVENT_SYSTEM_DRAGDROPSTART	No equivalent
EVENT_SYSTEM_FOREGROUND	AutomationFocusChangedEvent
EVENT_SYSTEM_MENUEND	MenuClosedEvent
EVENT_SYSTEM_MENUPOPUPEND	MenuClosedEvent
EVENT_SYSTEM_MENUPOPUPSTART	MenuOpenedEvent
EVENT_SYSTEM_MENUSTART	MenuOpenedEvent
EVENT_SYSTEM_MINIMIZEEND	WindowVisualStateProperty property change
EVENT_SYSTEM_MINIMIZESTART	WindowVisualStateProperty property change
EVENT_SYSTEM_MOVESIZEEND	BoundingRectangleProperty property change
EVENT_SYSTEM_MOVESIZESTART	BoundingRectangleProperty property change
EVENT_SYSTEM_SCROLLINGEND	VerticalScrollPercentProperty or HorizontalScrollPercentProperty property change

WINEVENT	UI AUTOMATION EVENT IDENTIFIER
EVENT_SYSTEM_SCROLLINGSTART	VerticalScrollPercentProperty or HorizontalScrollPercentProperty property change
EVENT_SYSTEM_SOUND	No equivalent
EVENT_SYSTEM_SWITCHEND	No equivalent, but an AutomationFocusChangedEvent event signals that a new application has received the focus
EVENT_SYSTEM_SWITCHSTART	No equivalent
No equivalent	CurrentViewProperty property change
No equivalent	HorizontallyScrollableProperty property change
No equivalent	VerticallyScrollableProperty property change
No equivalent	HorizontalScrollPercentProperty property change
No equivalent	VerticalScrollPercentProperty property change
No equivalent	HorizontalViewSizeProperty property change
No equivalent	VerticalViewSizeProperty property change
No equivalent	ToggleStateProperty property change
No equivalent	WindowVisualStateProperty property change
No equivalent	AsyncContentLoadedEvent event
No equivalent	ToolTipOpenedEvent

Security

Some `IAccessible` customization scenarios require wrapping a base `IAccessible` and calling through to it. This has security implications, since a partially trusted component should not be an intermediary on a code path.

The UI Automation model removes the need for providers to call through to other provider code. The UI Automation core service does all the necessary aggregation.

See also

- UI Automation Fundamentals

UI Automation Tree Overview

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Assistive technology products and test scripts navigate the UI Automation tree to gather information about the user interface (UI) and its elements.

Within the UI Automation tree there is a root element (RootElement) that represents the current desktop and whose child elements represent application windows. Each of these child elements can contain elements representing pieces of UI such as menus, buttons, toolbars, and list boxes. These elements in turn can contain elements such as list items.

The UI Automation tree is not a fixed structure and is seldom seen in its totality because it might contain thousands of elements. Parts of it are built as they are needed, and it can undergo changes as elements are added, moved, or removed.

UI Automation providers support the UI Automation tree by implementing navigation among items within a fragment, which consists of a root (usually hosted in a window) and a subtree. However, providers are not concerned with navigation from one control to another. This is managed by the UI Automation core, using information from the default window providers.

Views of the Automation Tree

The UI Automation tree can be filtered to create views that contain only those AutomationElement objects relevant for a particular client. This approach allows clients to customize the structure presented through UI Automation to their particular needs.

The client has two ways of customizing the view: by scoping and by filtering. Scoping is defining the extent of the view, starting from a base element: for example, the application might want to find only direct children of the desktop, or all descendants of an application window. Filtering is defining the types of elements that are to be included in the view.

UI Automation providers support filtering by defining properties on elements, including the IsControlElementProperty and IsContentElementProperty properties.

UI Automation provides three default views. These views are defined by the type of filtering performed; the scope of any view is defined by the application. In addition, the application can apply other filters on properties; for example, to include only enabled controls in a control view.

Raw View

The raw view of the UI Automation tree is the full tree of AutomationElement objects for which the desktop is the root. The raw view closely follows the native programmatic structure of an application and therefore is the most detailed view available. It is also the base on which the other views of the tree are built. Because this view depends on the underlying UI framework, the raw view of a WPF button will have a different raw view than a Win32 button.

The raw view is obtained by searching for elements without specifying properties or by using the RawViewWalker to navigate the tree.

Control View

The control view of the UI Automation tree simplifies the assistive technology product's task of describing the UI to the end user and helping that end user interact with the application because it closely maps to the UI structure perceived by an end user.

The control view is a subset of the raw view. It includes all UI items from the raw view that an end user would understand as interactive or contributing to the logical structure of the control in the UI. Examples of UI items that contribute to the logical structure of the UI, but are not interactive themselves, are item containers such as list view headers, toolbars, menus, and the status bar. Non-interactive items used simply for layout or decorative purposes will not be seen in the control view. An example is a panel that was used only to lay out the controls in a dialog but does not itself contain any information. Non-interactive items that will be seen in the control view are graphics with information and static text in a dialog. Non-interactive items that are included in the control view cannot receive keyboard focus.

The control view is obtained by searching for elements that have the IsControlElement property set to `true`, or by using the ControlViewWalker to navigate the tree.

Content View

The content view of the UI Automation tree is a subset of the control view. It contains UI items that convey the true information in a user interface, including UI items that can receive keyboard focus and some text that is not a label on a UI item. For example, the values in a drop-down combo box will appear in the content view because they represent the information being used by an end user. In the content view, a combo box and list box are both represented as a collection of UI items where one, or perhaps more than one, item can be selected. The fact that one is always open and one can expand and collapse is irrelevant in the content view because it is designed to show the data, or content, that is being presented to the user.

The content view is obtained by searching for elements that have the IsContentElement property set to `true`, or by using the ContentViewWalker to navigate the tree.

See also

- AutomationElement
- UI Automation Overview

UI Automation Control Patterns Overview

3/12/2020 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This overview introduces Microsoft UI Automation control patterns. Control patterns provide a way to categorize and expose a control's functionality independent of the control type or the appearance of the control.

UI Automation uses control patterns to represent common control behaviors. For example, you use the Invoke control pattern for controls that can be invoked (such as buttons) and the Scroll control pattern for controls that have scroll bars (such as list boxes, list views, or combo boxes). Because each control pattern represents a separate functionality, they can be combined to describe the full set of functionality supported by a particular control.

NOTE

Aggregate controls—built with child controls that provide the user interface (UI) for functionality exposed by the parent—should implement all control patterns normally associated with each child control. In turn, those same control patterns are not required to be implemented by the child controls.

UI Automation Control Pattern Components

Control patterns support the methods, properties, events, and relationships needed to define a discrete piece of functionality available in a control.

- The relationship between a UI Automation element and its parent, children and siblings describes the element's structure within the UI Automation tree.
- The methods allow UI Automation clients to manipulate the control.
- The properties and events provide information about the control pattern's functionality as well as information about the state of the control.

Control patterns relate to UI as interfaces relate to Component Object Model (COM) objects. In COM, you can query an object to ask what interfaces it supports and then use those interfaces to access functionality. In UI Automation, UI Automation clients can ask a control which control patterns it supports and then interact with the control through the properties, methods, events, and structures exposed by the supported control patterns. For example, for a multiline edit box, UI Automation providers implement IScrollProvider. When a client knows that an AutomationElement supports the ScrollPattern control pattern, it can use the properties, methods, and events exposed by that control pattern to manipulate the control, or access information about the control.

UI Automation Providers and Clients

UI Automation providers implement control patterns to expose the appropriate behavior for a specific piece of functionality supported by the control.

UI Automation clients access methods and properties of UI Automation control pattern classes and use them to get information about the UI, or to manipulate the UI. These control pattern classes are found in the System.Windows.Automation namespace (for example, InvokePattern and SelectionPattern).

Clients use AutomationElement methods (such as AutomationElement.GetCurrentPropertyValue or AutomationElement.GetCachedPropertyValue) or the common language runtime (CLR) accessors to access the UI Automation properties on a pattern. Each control pattern class has a field member (for example, InvokePattern.Pattern or SelectionPattern.Pattern) that identifies that control pattern and can be passed as a parameter to GetCachedPattern or GetCurrentPattern to retrieve that pattern for an AutomationElement.

Dynamic Control Patterns

Some controls do not always support the same set of control patterns. Control patterns are considered supported when they are available to a UI Automation client. For example, a multiline edit box enables vertical scrolling only when it contains more lines of text than can be displayed in its viewable area. Scrolling is disabled when enough text is removed so that scrolling is no longer required. For this example, the ScrollPattern control pattern is dynamically supported depending on the current state of the control (how much text is in the edit box).

Control Pattern Classes and Interfaces

The following table describes the UI Automation control patterns. The table also lists the classes used by UI Automation clients to access the control patterns, as well as the interfaces used by UI Automation providers to implement them.

CONTROL PATTERN CLASS	PROVIDER INTERFACE	DESCRIPTION
DockPattern	IDockProvider	Used for controls that can be docked in a docking container. For example, toolbars or tool palettes.
ExpandCollapsePattern	IExpandCollapseProvider	Used for controls that can be expanded or collapsed. For example, menu items in an application such as the **File** menu.
GridPattern	IGridProvider	Used for controls that support grid functionality such as sizing and moving to a specified cell. For example, the large icon view in Windows Explorer or simple tables without headers in Microsoft Word.
GridItemPattern	IGridItemProvider	Used for controls that have cells within grids. The individual cells should support the GridItem pattern. For example, each cell in Microsoft Windows Explorer detail view.
InvokePattern	IInvokeProvider	Used for controls that can be invoked, such as a button.

CONTROL PATTERN CLASS	PROVIDER INTERFACE	DESCRIPTION
MultipleViewPattern	IMultipleViewProvider	Used for controls that can switch between multiple representations of the same set of information, data, or children. For example, a list view control where data is available in thumbnail, tile, icon, list, or detail views.
RangeValuePattern	IRangeValueProvider	Used for controls that have a range of values that can be applied to the control. For example, a spinner control containing years might have a range of 1900 to 2010, while another spinner control presenting months would have a range of 1 to 12.
ScrollPattern	IScrollProvider	Used for controls that can scroll. For example, a control that has scroll bars that are active when there is more information than can be displayed in the viewable area of the control.
ScrollItemPattern	IScrollItemProvider	Used for controls that have individual items in a list that scrolls. For example, a list control that has individual items in the scroll list, such as a combo box control.
SelectionPattern	ISelectionProvider	Used for selection container controls. For example, list boxes and combo boxes.
SelectionItemPattern	ISelectionItemProvider	Used for individual items in selection container controls, such as list boxes and combo boxes.
TablePattern	ITableProvider	Used for controls that have a grid as well as header information. For example, Microsoft Excel worksheets.
TableItemPattern	ITableItemProvider	Used for items in a table.
TextPattern	ITextProvider	Used for edit controls and documents that expose textual information.
TogglePattern	IToggleProvider	Used for controls where the state can be toggled. For example, check boxes and checkable menu items.

CONTROL PATTERN CLASS	PROVIDER INTERFACE	DESCRIPTION
TransformPattern	ITransformProvider	Used for controls that can be resized, moved, and rotated. Typical uses for the Transform control pattern are in designers, forms, graphical editors, and drawing applications.
ValuePattern	IValueProvider	Allows clients to get or set a value on controls that do not support a range of values. For example, a date time picker.
WindowPattern	IWindowProvider	Exposes information specific to windows, a fundamental concept to the Microsoft Windows operating system. Examples of controls that are windows are top-level application windows (Microsoft Word, Microsoft Windows Explorer, and so on), multiple-document interface (MDI) child windows, and dialogs.

See also

- UI Automation Control Patterns for Clients
- Control Pattern Mapping for UI Automation Clients
- UI Automation Overview
- UI Automation Properties for Clients
- UI Automation Events for Clients

UI Automation Properties Overview

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

UI Automation providers expose properties on Microsoft UI Automation elements. These properties enable UI Automation client applications to discover information about pieces of the user interface (UI), especially controls, including both static and dynamic data.

This section gives a broad overview of Microsoft UI Automation properties. More specific information is given in the following topics:

- UI Automation Properties for Clients
- Server-Side UI Automation Provider Implementation

Property Identifiers

Every property is identified by a number and a name. The names of properties are used only for debugging and diagnosis. Providers use the numeric IDs to identify incoming property requests. Client applications, however, only use AutomationProperty, which encapsulates the number and name, to identify properties they wish to retrieve.

AutomationProperty objects representing particular properties are available as fields in various classes. For security reasons, UI Automation providers obtain these objects from a separate set of classes that are contained in Uiautomationtypes.dll.

The following table categorizes properties by the classes that contain the AutomationPropertyIDs.

KINDS OF PROPERTIES	CLIENTS GET IDS FROM	PROVIDERS GET IDS FROM
Properties common to all elements (see following tables)	AutomationElement	AutomationElementIdentifiers
Position of a docking window	DockPattern	DockPatternIdentifiers
State of an element that can expand and collapse	ExpandCollapsePattern	ExpandCollapsePatternIdentifiers
Properties of an item in a grid	GridItemPattern	GridItemPatternIdentifiers
Properties of a grid	GridPattern	GridPatternIdentifiers
Current and supported view of an element that has multiple views	MultipleViewPattern	MultipleViewPatternIdentifiers
Properties of an element that moves over a range of values, such as a slider	RangeValuePattern	RangeValuePatternIdentifiers

KINDS OF PROPERTIES	CLIENTS GET IDS FROM	PROVIDERS GET IDS FROM
Properties of a scrolling window	ScrollPattern	ScrollPatternIdentifiers
Status and container of an item that can be selected, as in a list	SelectionItemPattern	SelectionItemPatternIdentifiers
Properties of a control that contains selection items	SelectionPattern	SelectionPatternIdentifiers
Column and row headers of an item in a table	TableItemPattern	TableItemPatternIdentifiers
Column and row headers, and orientation, of a table	TablePattern	TablePatternIdentifiers
State of a toggle control	TogglePattern	TogglePatternIdentifiers
Capabilities of an element that can be moved, rotated, or resized	TransformPattern	TransformPatternIdentifiers
Value and read/write capabilities of an element that has a value	ValuePattern	ValuePatternIdentifiers
Capabilities and state of a window	WindowPattern	WindowPatternIdentifiers

Properties by Category

The following tables categorize the properties whose IDs are found in AutomationElement and AutomationElementIdentifiers. These properties are common to all controls. All but a few of them are likely to be static over the lifetime of the provider application; most dynamic properties are associated with control patterns.

The **Property Access** column lists any other accessors for each property, in addition to GetCurrentPropertyValue and GetCachedPropertyValue. For more information on getting properties in a client application, see UI Automation Properties for Clients.

NOTE
For specific information about each property, follow the link in the **Property Access** column.

Display Characteristics

PROPERTY IDENTIFIER	PROPERTY ACCESS
BoundingRectangleProperty	BoundingRectangle
CultureProperty	n/a
HelpTextProperty	HelpText
IsOffscreenProperty	IsOffscreen
OrientationProperty	Orientation

Element Type

PROPERTY IDENTIFIER	PROPERTY ACCESS
ControlTypeProperty	ControlType
IsContentElementProperty	IsContentElement
IsControlElementProperty	IsControlElement
ItemTypeProperty	ItemType
LocalizedControlTypeProperty	LocalizedControlType

Identification

PROPERTY IDENTIFIER	PROPERTY ACCESS
AutomationIdProperty	AutomationId
ClassNameProperty	ClassName
FrameworkIdProperty	FrameworkId
LabeledByProperty	LabeledBy
NameProperty	Name
ProcessIdProperty	ProcessId
RuntimeIdProperty	GetRuntimeId
NativeWindowHandleProperty	NativeWindowHandle

Interaction

PROPERTY IDENTIFIER	PROPERTY ACCESS
AcceleratorKeyProperty	AcceleratorKey
AccessKeyProperty	AccessKey
ClickablePointProperty	GetClickablePoint
HasKeyboardFocusProperty	HasKeyboardFocus
IsEnabledProperty	IsEnabled
IsKeyboardFocusableProperty	IsKeyboardFocusable

Support for Patterns

PROPERTY IDENTIFIER	PROPERTY ACCESS
IsDockPatternAvailableProperty	GetSupportedPatterns
IsExpandCollapsePatternAvailableProperty	GetSupportedPatterns
IsGridItemPatternAvailableProperty	GetSupportedPatterns
IsGridPatternAvailableProperty	GetSupportedPatterns
IsInvokePatternAvailableProperty	GetSupportedPatterns
IsMultipleViewPatternAvailableProperty	GetSupportedPatterns
IsRangeValuePatternAvailableProperty	GetSupportedPatterns
IsScrollItemPatternAvailableProperty	GetSupportedPatterns
IsScrollPatternAvailableProperty	GetSupportedPatterns
IsSelectionItemPatternAvailableProperty	GetSupportedPatterns
IsSelectionPatternAvailableProperty	GetSupportedPatterns
IsTableItemPatternAvailableProperty	GetSupportedPatterns
IsTablePatternAvailableProperty	GetSupportedPatterns
IsTextPatternAvailableProperty	GetSupportedPatterns
IsTogglePatternAvailableProperty	GetSupportedPatterns
IsTransformPatternAvailableProperty	GetSupportedPatterns
IsValuePatternAvailableProperty	GetSupportedPatterns
IsWindowPatternAvailableProperty	GetSupportedPatterns

Miscellaneous

PROPERTY IDENTIFIER	PROPERTY ACCESS
IsRequiredForFormProperty	IsRequiredForForm
IsPasswordProperty	IsPassword
ItemStatusProperty	ItemStatus

Localization

UI Automation providers should present the following properties in the language of the operating system:

- AcceleratorKeyProperty

- AccessKeyProperty
- HelpTextProperty
- LocalizedControlTypeProperty
- NameProperty

Properties and Events

Closely tied in with the properties in UI Automation is the concept of property-changed events. For dynamic properties, the client application needs a way to know that a property value has changed, so that it can update its cache of information or react to the new information in some other way.

Providers raise events when something in the UI changes. For example, if a check box is selected or cleared, a property-changed event is raised by the provider's implementation of the Toggle pattern. Providers can raise events selectively, depending on whether any clients are listening for events, or listening for specific events.

Not all property changes raise events; that is entirely up to the implementation of the UI Automation provider for the element. For example, the standard proxy providers for list boxes do not raise an event when the SelectionProperty changes. In this case, the application instead must listen for an ElementSelectedEvent.

Clients listen for events by subscribing to them. Subscribing to events means creating delegate methods that can handle the events, and then passing the methods to UI Automation along with the specific events that will be dealt with in those methods. For property-changed events in particular, clients must implement AutomationPropertyChangedEventHandler.

See also

- Caching in UI Automation Clients
- UI Automation Properties for Clients
- Server-Side UI Automation Provider Implementation
- Find a UI Automation Element Based on a Property Condition
- Return Properties from a UI Automation Provider
- Raise Events from a UI Automation Provider

UI Automation Events Overview

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Microsoft UI Automation event notification is a key feature for assistive technologies such as screen readers and screen magnifiers. These UI Automation clients track events that are raised by UI Automation providers when something happens in the UI and use the information to notify end users.

Efficiency is improved by allowing provider applications to raise events selectively, depending on whether any clients are subscribed to those events, or not at all, if no clients are listening for any events.

Types of Events

UI Automation events fall into the following categories.

EVENT	DESCRIPTION
Property change	Raised when a property on an UI Automation element or control pattern changes. For example, if a client needs to monitor an application's check box control, it can register to listen for a property change event on the ToggleState property. When the check box control is checked or unchecked, the provider raises the event and the client can act as necessary.
Element action	Raised when a change in the UI results from end user or programmatic activity; for example, when a button is clicked or invoked through InvokePattern.
Structure change	Raised when the structure of the UI Automation tree changes. The structure changes when new UI items become visible, hidden, or removed on the desktop.
Global desktop change	Raised when actions of global interest to the client occur, such as when the focus shifts from one element to another, or when a window closes.

Some events do not necessarily mean that the state of the UI has changed. For example, if the user tabs to a text entry field and then clicks a button to update the field, a `TextChangedEvent` is raised even if the user did not actually change the text. When processing an event, it may be necessary for a client application to check whether anything has actually changed before taking action.

The following events may be raised even when the state of the UI has not changed.

- `AutomationPropertyChangedEvent` (depending on the property that has changed)
- `ElementSelectedEvent`

- `InvalidatedEvent`
- `TextChangedEvent`

UI Automation Event Identifiers

Microsoft UI Automation events are identified by AutomationEvent objects. The Id property contains a value that uniquely identifies the kind of event.

The possible values for Id are given in the following table, along with the type used for event arguments. Note that the identifiers used by clients and providers are identically named fields from different classes.

CLIENT IDENTIFIER	PROVIDER IDENTIFIER	EVENT ARGUMENTS TYPE
AutomationElement.AsyncContentLoadedEvent	AutomationElementIdentifiers.AsyncContentLoadedEvent	AsyncContentLoadedEventArgs
SelectionItemPattern.ElementAddedToSelectionEvent SelectionItemPattern.ElementRemovedFromSelectionEvent SelectionItemPattern.ElementSelectedEvent SelectionPattern.InvalidatedEvent InvokePattern.InvokedEvent AutomationElement.LayoutInvalidatedEvent AutomationElement.MenuClosedEvent AutomationElement.MenuOpenedEvent TextPattern.TextChangedEvent TextPattern.TextSelectionChangedEvent AutomationElement.ToolTipClosedEvent AutomationElement.ToolTipOpenedEvent WindowPattern.WindowOpenedEvent	SelectionItemPatternIdentifiers.ElementAddedToSelectionEvent SelectionItemPatternIdentifiers.ElementRemovedFromSelectionEvent SelectionItemPatternIdentifiers.ElementSelectedEvent SelectionPatternIdentifiers.InvalidatedEvent InvokePatternIdentifiers.InvokedEvent AutomationElementIdentifiers.LayoutInvalidatedEvent AutomationElementIdentifiers.MenuClosedEvent AutomationElementIdentifiers.MenuOpenedEvent TextPatternIdentifiers.TextChangedEvent TextPatternIdentifiers.TextSelectionChangedEvent AutomationElementIdentifiers.ToolTipClosedEvent AutomationElementIdentifiers.ToolTipOpenedEvent WindowPatternIdentifiers.WindowOpenedEvent	AutomationEventArgs
AutomationElement.AutomationFocusChangedEvent	AutomationElementIdentifiers.AutomationFocusChangedEvent	AutomationFocusChangedEventArgs
AutomationElement.AutomationPropertyChangedEvent	AutomationElementIdentifiers.AutomationPropertyChangedEvent	AutomationPropertyChangedEventArgs

CLIENT IDENTIFIER	PROVIDER IDENTIFIER	EVENT ARGUMENTS TYPE
AutomationElement.StructureChangedEvent	AutomationElementIdentifiers.StructureChangedEvent	StructureChangedEventArgs
WindowPattern.WindowClosedEvent	WindowPatternIdentifiers.WindowClosedEvent	WindowClosedEventArgs

UI Automation Event Arguments

The following classes encapsulate event arguments.

CLASS	DESCRIPTION
AsyncContentLoadedEventArgs	Contains information about the asynchronous loading of content, including the percentage of loading completed.
AutomationEventArgs	Contains information about a simple event that requires no extra data.
AutomationFocusChangedEventArgs	Contains information about a change in input focus from one element to another. Events of this type are raised by the UI Automation system, not by providers.
AutomationPropertyChangedEventArgs	Contains information about a change in a property value of an element or control pattern.
StructureChangedEventArgs	Contains information about a change in the UI Automation tree.
WindowClosedEventArgs	Contains information about a window closing.

All the event argument classes contain an EventId member. This identifier is encapsulated in an AutomationEvent.

The AutomationEvent objects used to identify events are obtained by providers from fields in AutomationElementIdentifiers and control pattern identifier classes such as DockPatternIdentifiers. The equivalent fields are obtained by client applications from fields in AutomationElement and control pattern classes such as DockPattern.

For a list of event identifiers, see UI Automation Events for Clients.

See also

- UI Automation Events for Clients
- Server-Side UI Automation Provider Implementation
- Subscribe to UI Automation Events

UI Automation Security Overview

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This overview describes the security model for Microsoft UI Automation in Windows Vista.

User Account Control

Security is a major focus of Windows Vista and among the innovations is the ability for users to run as standard (non-administrator) users without necessarily being blocked from running applications and services that require higher privileges.

In Windows Vista, most applications are supplied with either a standard or an administrative token. If an application cannot be identified as an administrative application, it is launched as a standard application by default. Before an application identified as administrative can be launched, Windows Vista prompts the user for consent to run the application as elevated. The consent prompt is displayed by default, even if the user is a member of the local Administrators group, because administrators run as standard users until an application or system component that requires administrative credentials requests permission to run.

Tasks Requiring Higher Privileges

When a user attempts to perform a task that requires administrative privileges, Windows Vista presents a dialog box asking the user for consent to continue. This dialog box is protected from cross-process communication, so that malicious software cannot simulate user input. Similarly, the desktop logon screen cannot normally be accessed by other processes.

UI Automation clients must communicate with other processes, some of them perhaps running at a higher privilege level. Clients also might need access to the system dialog boxes that are not normally visible to other processes. Therefore, UI Automation clients must be trusted by the system, and must run with special privileges.

To be trusted to communicate with applications running at a higher privilege level, applications must be signed.

Manifest Files

To gain access to the protected system UI, applications must be built with a manifest file that includes the `uiAccess` attribute in the `requestedExecutionLevel` tag, as follows:

```
<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
  <security>
    <requestedPrivileges>
      <requestedExecutionLevel
        level="highestAvailable"
        uiAccess="true" />
    </requestedPrivileges>
  </security>
</trustInfo>
```

The value of the `level` attribute in this code is an example only.

`uiAccess` is "false" by default; that is, if the attribute is omitted, or if there is no manifest for the assembly, the application will not be able to gain access to protected UI.

Using UI Automation for Automated Testing

1/28/2020 • 7 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This overview describes how Microsoft UI Automation can be useful as a framework for programmatic access in automated testing scenarios.

UI Automation provides a unified object model that enables all user interface (UI) frameworks to expose complex and rich functionality in an accessible and easily automated manner.

UI Automation was developed as a successor to Microsoft Active Accessibility. Active Accessibility is an existing framework designed to provide a solution for making controls and applications accessible. Active Accessibility was not designed with test automation in mind even though it evolved into that role due to the very similar requirements of accessibility and automation. UI Automation, in addition to providing more refined solutions for accessibility, is also specifically designed to provide robust functionality for automated testing. For example, Active Accessibility relies on a single interface to both expose information about the UI and collect the information needed by AT products; UI Automation separates the two models.

Both a provider and client are required to implement UI Automation for it to be useful as an automated test tool. UI Automation providers are applications such as Microsoft Word, Excel, and other third-party applications or controls based on the Microsoft Windows operating system. UI Automation clients include automated test scripts and assistive technology applications.

NOTE

The intent of this overview is to showcase the new and improved automated testing capabilities of UI Automation. This overview is not intended to provide information on accessibility features and will not address accessibility other than where necessary.

UI Automation in a Provider

For a UI to be automated, a developer of an application or control must look at what actions an end-user can perform on the UI object using standard keyboard and mouse interaction.

Once these key actions have been identified, the corresponding UI Automation control patterns (that is, the control patterns that mirror the functionality and behavior of the UI element) should be implemented on the control. For example, user interaction with a combo box control (such as the run dialog) typically involves expanding and collapsing the combo box to hide or display a list of items, selecting an item from that list, or adding a new value via keyboard input.

NOTE

With other accessibility models, developers must gather information directly from individual buttons, menus, or other controls. Unfortunately, every control type comes in dozens of minor variations. In other words, even though ten variations of a pushbutton may all work the same way and perform the same function, they must all be treated as unique controls. There is no way to know that these controls are functionally equivalent. Control patterns were developed to represent these common control behaviors. For more information, see UI Automation Control Patterns Overview.

Implementing UI Automation

As mentioned earlier, without the unified model provided by UI Automation, test tools and developers are required to know framework-specific information in order to expose properties and behaviors of controls in that framework. Since there can be several different UI frameworks present at any single time within Windows operating systems, including Win32, Windows Forms, and Windows Presentation Foundation (WPF), it can be a daunting task to test multiple applications with controls that seem similar. For example, the following table outlines the framework-specific property names required to retrieve the name (or text) associated with a button control and shows the single equivalent UI Automation property.

UI AUTOMATION CONTROL TYPE	UI FRAMEWORK	FRAMEWORK SPECIFIC PROPERTY	UI AUTOMATION PROPERTY
Button	Windows Presentation Foundation	Content	NameProperty
Button	Win32	Caption	NameProperty
Image	HTML	alt	NameProperty

UI Automation providers are responsible for mapping the framework-specific properties of their controls to the equivalent UI Automation properties.

Information on implementing UI Automation in a provider can be found at UI Automation Providers for Managed Code. Information on implementing control patterns is available at UI Automation Control Patterns and UI Automation Text Pattern.

UI Automation in a Client

The goal of many automated test tools and scenarios is the consistent and repeatable manipulation of the user interface. This can involve unit testing specific controls through to the recording and playback of test scripts that iterate through a series of generic actions on a group of controls.

A complication that arises from automated applications is the difficulty synchronizing a test with a dynamic target. For example, a list box control, such as one contained in the Windows Task Manager, that displays a list of currently running applications. Since the items in the list box are dynamically updated outside the control of the test application, attempting to repeat the selection of a specific item in the list box with any consistency is impossible. Similar issues can also arise when attempting to repeat simple focus changes in a UI that is outside the control of the test application.

Programmatic Access

Programmatic access provides the ability to imitate, through code, any interaction and experience exposed by traditional mouse and keyboard input. UI Automation enables programmatic access through five components:

- The UI Automation tree facilitates navigation through the structure of the UI. The tree is built from the collection of hWnd's. For more information, see UI Automation Tree Overview
- Automation elements are individual components in the UI. These can often be more granular than an hWnd.

For more information, see UI Automation Control Types Overview.

- Automation properties provide specific information about UI elements. For more information, see UI Automation Properties Overview.
- Control patterns define a particular aspect of a control's functionality; they can consist of property, method, event, and structure information. For more information, see UI Automation Control Patterns Overview.
- Automation events provide event notifications and information. For more information, see UI Automation Events Overview.

Key Properties for Test Automation

The ability to uniquely identify and subsequently locate any control within the UI provides the basis for automated test applications to operate on that UI. There are several Microsoft UI Automation properties used by clients and providers that assist in this.

AutomationID

Uniquely identifies an automation element from its siblings. AutomationIdProperty is not localized, unlike a property such as NameProperty that is typically localized if a product gets shipped in multiple languages. See Use the AutomationID Property.

> **NOTE**
>
> AutomationIdProperty does not guarantee a unique identity throughout the automation tree. For example, an application may contain a menu control with multiple top-level menu items that, in turn, have multiple child menu items. These secondary menu items may be identified by a generic scheme such as "Item1, Item 2, Item3, etc.", allowing duplicate identifiers for children across top-level menu items.

ControlType

Identifies the type of control represented by an automation element. Significant information can be inferred from knowledge of the control type. See UI Automation Control Types Overview.

NameProperty

This is a text string that identifies or explains a control. NameProperty should be used with caution since it can be localized. See UI Automation Properties Overview.

Implementing UI Automation in a Test Application

Add the UI Automation References.	The UI Automation dll's necessary for UI Automation clients are listed here. - UIAutomationClient.dll provides access to the UI Automation client-side APIs. - UIAutomationClientSideProvider.dll provides the ability to automate Win32 controls. See UI Automation Support for Standard Controls. - UIAutomationTypes.dll provides access to the specific types defined in UI Automation.
Add the System.Windows.Automation namespace.	This namespace contains everything UI Automation clients need to use the capabilities of UI Automation except text handling.
Add the System.Windows.Automation.Text namespace.	This namespace contains everything a UI Automation clients need to use the capabilities of UI Automation text handling.

Find controls of interest	Automated test scripts locate UI Automation elements that represent controls of interest within the automation tree. There are multiple ways to obtain UI Automation elements with code. - Query the UI using a Condition statement. This is typically where the language-neutral AutomationIdProperty is used. **Note:** An AutomationIdProperty can be obtained using a tool such as Inspect.exe that is able to itemize the UI Automation properties of a control. - Use the TreeWalker class to traverse the entire UI Automation tree or a subset thereof. - Track focus. - Use the hWnd of the control. - Use screen location, such as the location of the mouse cursor. See Obtaining UI Automation Elements
Obtain Control Patterns	Control patterns expose common behaviors for functionally similar controls. After locating the controls that require testing, automated test scripts obtain the control patterns of interest from those UI Automation elements. For example, the InvokePattern control pattern for typical button functionality or the WindowPattern control pattern for window functionality. See UI Automation Control Patterns Overview.
Automate the UI	Automated test scripts can now control any UI of interest from a UI framework using the information and functionality exposed by the UI Automation control patterns.

Related Tools and Technologies

There are a number of related tools and technologies that support automated testing with UI Automation.

- Inspect.exe is a graphical user interface (GUI) application that can be used to gather UI Automation information for both provider and client development and debugging. Inspect.exe is included in the Windows SDK.
- MSAABridge exposes UI Automation information to Active Accessibility clients. The primary goal of bridging UI Automation to Active Accessibility is to allow existing Active Accessibility clients the ability to interact with any framework that has implemented UI Automation.

Security

For security information, see UI Automation Security Overview.

See also

- UI Automation Fundamentals

UI Automation Providers for Managed Code

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section contains overviews and how-to topics that describe how to write Microsoft UI Automation providers for custom user interface (UI) elements.

In This Section

Server-Side UI Automation Provider Implementation
UI Automation Providers Overview
Client-Side UI Automation Provider Implementation
How-to Topics

Server-Side UI Automation Provider Implementation

1/28/2020 • 8 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section describes how to implement a server-side UI Automation provider for a custom control.

The implementation for Windows Presentation Foundation (WPF) elements and non-WPF elements (such as those designed for Windows Forms) is fundamentally different. WPF elements provide support for UI Automation through a class derived from AutomationPeer. Non-WPF elements provide support through implementations of provider interfaces.

Security Considerations

Providers should be written so that they can work in a partial-trust environment. Because UIAutomationClient.dll is not configured to run under partial trust, your provider code should not reference that assembly. If it does so, the code may run in a full-trust environment but then fail in a partial-trust environment.

In particular, do not use fields from classes in UIAutomationClient.dll such as those in AutomationElement. Instead, use the equivalent fields from classes in UIAutomationTypes.dll, such as AutomationElementIdentifiers.

Provider Implementation by Windows Presentation Foundation Elements

For more information on this topic, please see UI Automation of a WPF Custom Control.

Provider Implementation by non-WPF Elements

Custom controls that are not part of the WPF framework, but that are written in managed code (most often these are Windows Forms controls), provide support for UI Automation by implementing interfaces. Every element must implement at least one of the interfaces listed in the first table in the next section. In addition, if the element supports one or more control patterns, it must implement the appropriate interface for each control pattern.

Your UI Automation provider project must reference the following assemblies:

- UIAutomationProviders.dll
- UIAutomationTypes.dll
- WindowsBase.dll

Provider Interfaces

Every UI Automation provider must implement one of the following interfaces.

INTERFACE	DESCRIPTION

INTERFACE	DESCRIPTION
IRawElementProviderSimple	Provides functionality for a simple control hosted in a window, including support for control patterns and properties.
IRawElementProviderFragment	Inherits from IRawElementProviderSimple. Adds functionality for an element in a complex control, including navigation within the fragment, setting focus, and returning the bounding rectangle of the element.
IRawElementProviderFragmentRoot	Inherits from IRawElementProviderFragment. Adds functionality for the root element in a complex control, including locating a child element at specified coordinates and setting the focus state for the entire control.

The following interfaces provide added functionality but are not required to be implemented.

INTERFACE	DESCRIPTION
IRawElementProviderAdviseEvents	Enables the provider to track requests for events.
IRawElementProviderHwndOverride	Enables repositioning of window-based elements within the UI Automation tree of a fragment.

All other interfaces in the System.Windows.Automation.Provider namespace are for control pattern support.

Requirements for Non-WPF Providers

In order to communicate with UI Automation, your control must implement the following main areas of functionality:

FUNCTIONALITY	IMPLEMENTATION
Expose the provider to UI Automation	In response to a WM_GETOBJECT message sent to the control window, return the object that implements IRawElementProviderSimple (or a derived interface). For fragments, this must be the provider for the fragment root.
Provide property values	Implement GetPropertyValue to provide or override values.
Enable the client to interact with the control	Implement interfaces that support control patterns, such as IInvokeProvider. Return these pattern providers in your implementation of GetPatternProvider.
Raise events	Call one of the static methods of AutomationInteropProvider to raise an event that a client can listen for.
Enable navigation and focusing within a fragment	Implement IRawElementProviderFragment for each element within the fragment. (Not necessary for elements that are not part of a fragment.)
Enable focusing and location of child element in a fragment	Implement IRawElementProviderFragmentRoot. (Not necessary for elements that are not fragment roots.)

Property Values in Non-WPF Providers

UI Automation providers for custom controls must support certain properties that can be used by the automation system as well as by client applications. For elements that are hosted in windows (HWNDs), UI Automation can retrieve some properties from the default window provider, but must obtain others from the custom provider.

Providers for HWND based controls do not usually need to provide the following properties (identified by field values):

- BoundingRectangleProperty
- ClickablePointProperty
- ProcessIdProperty
- ClassNameProperty
- HasKeyboardFocusProperty
- IsEnabledProperty
- IsKeyboardFocusableProperty
- IsPasswordProperty
- NameProperty
- RuntimeIdProperty

NOTE

The RuntimeIdProperty of a simple element or fragment root hosted in a window is obtained from the window; however, fragment elements below the root (such as list items in a list box) must provide their own identifiers. For more information, see GetRuntimeId.

The IsKeyboardFocusableProperty should be returned for providers hosted in a Windows Forms control. In this case, the default window provider may be unable to retrieve the correct value.

The NameProperty is usually supplied by the host provider. For example, if a custom control is derived from Control, the name is derived from the `Text` property of the control.

For example code, see Return Properties from a UI Automation Provider.

Events in Non-WPF Providers

UI Automation providers should raise events to notify client applications of changes in the state of the UI. The following methods are used to raise events.

METHOD	DESCRIPTION
RaiseAutomationEvent	Raises various events, including events triggered by control patterns.
RaiseAutomationPropertyChangedEvent	Raises an event when a UI Automation property has changed.
RaiseStructureChangedEvent	Raises an event when the structure of the UI Automation tree has changed; for example, by the removal or addition of an element.

The purpose of an event is to notify the client of something taking place in the user interface (UI), whether or not the activity is triggered by the UI Automation system itself. For example, the event identified by InvokedEvent should be raised whenever the control is invoked, either through direct user input or by the client application calling Invoke.

To optimize performance, a provider can selectively raise events, or raise no events at all if no client application is registered to receive them. The following methods are used for optimization.

METHOD	DESCRIPTION
ClientsAreListening	This static property specifies whether any client applications have subscribed to UI Automation events.
IRawElementProviderAdviseEvents	The provider's implementation of this interface on a fragment root enables it to be advised when clients register and unregister event handlers for events on the fragment.

Non-WPF Provider Navigation

Providers for simple controls such as a custom button hosted in a window (HWND) do not need to support navigation within the UI Automation tree. Navigation to and from the element is handled by the default provider for the host window, which is specified in the implementation of HostRawElementProvider. When you implement a provider for a complex custom control, however, you must support navigation between the root node of the fragment and its descendants, and between sibling nodes.

NOTE

Elements of a fragment other than the root must return a `null` reference from HostRawElementProvider, because they are not directly hosted in a window, and no default provider can support navigation to and from them.

The structure of the fragment is determined by your implementation of Navigate. For each possible direction from each fragment, this method returns the provider object for the element in that direction. If there is no element in that direction, the method returns a `null` reference.

The fragment root supports navigation only to child elements. For example, a list box returns the first item in the list when the direction is FirstChild, and the last item when the direction is LastChild. The fragment root does not support navigation to a parent or siblings; this is handled by the host window provider.

Elements of a fragment that are not the root must support navigation to the parent, and to any siblings and children they have.

Non-WPF Provider Reparenting

Pop-up windows are actually top-level windows, and so by default appear in the UI Automation tree as children of the desktop. In many cases, however, pop-up windows are logically children of some other control. For example, the drop-down list of a combo box is logically a child of the combo box. Similarly, a menu pop-up window is logically a child of the menu. UI Automation provides support to reparent pop-up windows so that they appear to be children of the associated control.

To reparent a pop-up window:

1. Create a provider for the pop-up window. This requires that the class of the pop-up window is known in advance.

2. Implement all properties and patterns as usual for that pop-up, as though it were a control in its own right.

3. Implement the HostRawElementProvider property so that it returns the value obtained from HostProviderFromHandle, where the parameter is the window handle of the pop-up window.

4. Implement Navigate for the pop-up window and its parent so that navigation is handled properly from the logical parent to the logical children, and between sibling children.

When UI Automation encounters the pop-up window, it recognizes that navigation is being overridden from the default, and skips over the pop-up window when it is encountered as a child of the desktop. Instead, the node will only be reachable through the fragment.

Reparenting is not suitable for cases where a control can host a window of any class. For example, a rebar can host any type of HWND in its bands. To handle these cases, UI Automation supports an alternative form of HWND relocation, as described in the next section.

Non-WPF Provider Repositioning

UI Automation fragments may contain two or more elements that are each contained in a window (HWND). Because each HWND has its own default provider that considers the HWND to be a child of a containing HWND, the UI Automation tree will, by default, show the HWNDs in the fragment as children of the parent window. In most cases this is desirable behavior, but sometimes it can lead to confusion because it does not match the logical structure of the UI.

A good example of this is a rebar control. A rebar contains bands, each of which can in turn contain an HWND-based control such as a toolbar, an edit box, or a combo box. The default window provider for the rebar HWND sees the band control HWNDs as children, and the rebar provider sees the bands as children. Because the HWND provider and the rebar provider are working in tandem and combining their children, both the bands and the HWND-based controls appear as children of the rebar. Logically, however, only the bands should appear as children of the rebar, and each band provider should be coupled with the default HWND provider for the control it contains.

To accomplish this, the fragment root provider for the rebar exposes a set of children representing the bands. Each band has a single provider that may expose properties and patterns. In its implementation of HostRawElementProvider, the band provider returns the default window provider for the control HWND, which it obtains by calling HostProviderFromHandle, passing in the control's window handle. Finally, the fragment root provider for the rebar implements the IRawElementProviderHwndOverride interface, and in its implementation of GetOverrideProviderForHwnd it returns the appropriate band provider for the control contained in the specified HWND.

See also

- UI Automation Providers Overview
- Expose a Server-side UI Automation Provider
- Return Properties from a UI Automation Provider
- Raise Events from a UI Automation Provider
- Enable Navigation in a UI Automation Fragment Provider
- Support Control Patterns in a UI Automation Provider

UI Automation Providers Overview

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

UI Automation providers enable controls to communicate with UI Automation client applications. In general, each control or other distinct element in a user interface (UI) is represented by a provider. The provider exposes information about the element and optionally implements control patterns that enable the client application to interact with the control.

Client applications do not usually have to work directly with providers. Most of the standard controls in applications that use the Win32, Windows Forms, or Windows Presentation Foundation (WPF) frameworks are automatically exposed to the UI Automation system. Applications that implement custom controls may also implement UI Automation providers for those controls, and client applications do not have to take any special steps to gain access to them.

This topic provides an overview of how control developers implement UI Automation providers, particularly for controls in Windows Forms and Win32 windows.

Types of Providers

UI Automation providers fall into two categories: client-side providers and server-side providers.

Client-side providers

Client-side providers are implemented by UI Automation clients to communicate with an application that does not support, or does not fully support, UI Automation. Client-side providers usually communicate with the server across the process boundary by sending and receiving Windows messages.

Because UI Automation providers for controls in Win32, Windows Forms, or WPF applications are supplied as part of the operating system, client applications seldom have to implement their own providers, and this overview does not cover them further.

Server-side providers

Server-side providers are implemented by custom controls or by applications that are based on a UI framework other than Win32, Windows Forms, or WPF.

Server-side providers communicate with client applications across the process boundary by exposing interfaces to the UI Automation core system, which in turn serves requests from clients.

UI Automation Provider Concepts

This section provides brief explanations of some of the key concepts you need to understand in order to implement UI Automation providers.

Elements

UI Automation elements are pieces of user interface (UI) that are visible to UI Automation clients. Examples include application windows, panes, buttons, tooltips, list boxes, and list items.

Navigation

UI Automation elements are exposed to clients as a UI Automation tree. UI Automation constructs the tree by navigating from one element to another. Navigation is enabled by the providers for each element, each of which may point to a parent, siblings, and children.

For more information on the client view of the UI Automation tree, see UI Automation Tree Overview.

Views

A client can see the UI Automation tree in three principal views, as shown in the following table.

Raw view	Contains all elements.
Control view	Contains elements that are controls.
Content view	Contains elements that have content.

For more information on client views of the UI Automation tree, see UI Automation Tree Overview.

It is the responsibility of the provider implementation to define an element as a content element or a control element. Control elements may or may not also be content elements, but all content elements are control elements.

Frameworks

A framework is a component that manages child controls, hit-testing, and rendering in an area of the screen. For example, a Win32 window, often referred to as an HWND, can serve as a framework that contains multiple UI Automation elements such as a menu bar, a status bar, and buttons.

Win32 container controls such as list boxes and tree views are considered to be frameworks, because they contain their own code for rendering child items and performing hit-testing on them. By contrast, a WPF list box is not a framework, because the rendering and hit-testing is being handled by the containing WPF window.

The UI in an application can be made up of different frameworks. For example, an HWND application window might contain Dynamic HTML (DHTML) which in turn contains a component such as a combo box in an HWND.

Fragments

A fragment is a complete subtree of elements from a particular framework. The element at the root node of the subtree is called a fragment root. A fragment root does not have a parent, but is hosted within some other framework, usually a Win32 window (HWND).

Hosts

The root node of every fragment must be hosted in an element, usually a Win32 window (HWND). The exception is the desktop, which is not hosted in any other element. The host of a custom control is the HWND of the control itself, not the application window or any other window that might contain groups of top-level controls.

The host of a fragment plays an important role in providing UI Automation services. It enables navigation to the fragment root, and supplies some default properties so that the custom provider does not have to implement them.

See also

- Server-Side UI Automation Provider Implementation

Client-Side UI Automation Provider Implementation

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Several different user interface (UI) frameworks are in use within Microsoft operating systems, including Win32, Windows Forms, and Windows Presentation Foundation (WPF). Microsoft UI Automation exposes information about UI elements to clients. However, UI Automation does not itself have awareness of the different types of controls that exist in these frameworks and the techniques that are needed to extract information from them. Instead, it leaves this task to objects called providers. A provider extracts information from a specific control and hands that information to UI Automation, which then presents it to the client in a consistent manner.

Providers can exist either on the server side or on the client side. A server-side provider is implemented by the control itself. WPF elements implement providers, as can any third-party controls written with UI Automation in mind.

However, older controls such as those in Win32 and Windows Forms do not directly support UI Automation. These controls are served instead by providers that exist in the client process and obtain information about controls using cross-process communication; for example, by monitoring windows messages to and from the controls. Such client-side providers are sometimes called proxies.

Windows Vista supplies providers for standard Win32 and Windows Forms controls. In addition, a fallback provider gives partial UI Automation support to any control that is not served by another server-side provider or proxy but has a Microsoft Active Accessibility implementation. All these providers are automatically loaded and available to client applications.

For more information on support for Win32 and Windows Forms controls, see UI Automation Support for Standard Controls.

Applications can also register other client-side providers.

Distributing Client-Side Providers

UI Automation expects to find client-side providers in a managed-code assembly. The namespace in this assembly should have the same name as the assembly. For example, an assembly called ContosoProxies.dll would contain the ContosoProxies namespace. Within the namespace, create a UIAutomationClientSideProviders class. In the implementation of the static ClientSideProviderDescriptionTable field, create an array of ClientSideProviderDescription structures describing the providers.

Registering and Configuring Client-Side Providers

Client-side providers in a dynamic-link library (DLL) are loaded by calling RegisterClientSideProviderAssembly. No further action is required by a client application to make use of the providers.

Providers implemented in the client's own code are registered by using RegisterClientSideProviders. This method takes as an argument an array of ClientSideProviderDescription structures, each of which specifies the following properties:

- A callback function that creates the provider object.
- The class name of the controls that the provider will serve.
- The image name of the application (usually the full name of the executable file) that the provider will serve.
- Flags that govern how the class name is matched against window classes found in the target application.

The last two parameters are optional. The client might specify the image name of the target application when it wants to use different providers for different applications. For example, the client might use one provider for a Win32 list view control in a known application that supports the Multiple View pattern, and another for a similar control in another known application that does not.

See also

- Create a Client-Side UI Automation Provider
- Implement UI Automation Providers in a Client Application

UI Automation Providers for Managed Code How-to Topics

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section contains code examples that demonstrate tasks in writing Microsoft UI Automation providers for user interface (UI) elements.

In This Section

Expose a Server-side UI Automation Provider
Return Properties from a UI Automation Provider
Raise Events from a UI Automation Provider
Enable Navigation in a UI Automation Fragment Provider
Support Control Patterns in a UI Automation Provider
Create a Client-Side UI Automation Provider
Implement UI Automation Providers in a Client Application

Expose a Server-side UI Automation Provider

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that shows how to expose a server-side UI Automation provider that is hosted in a System.Windows.Forms.Control window.

The example overrides the window procedure to trap WM_GETOBJECT, which is the message sent by the UI Automation core service when a client application requests information about the window.

Example

```
/// <summary>
/// Handles WM_GETOBJECT message; others are passed to base handler.
/// </summary>
/// <param name="m">Windows message.</param>
/// <remarks>
/// This method enables UI Automation to find the control.
/// In this example, the implementation of IRawElementProvider is in the same class
/// as this method.
/// </remarks>
protected override void WndProc(ref Message m)
{
    const int WM_GETOBJECT = 0x003D;

    if ((m.Msg == WM_GETOBJECT) && (m.LParam.ToInt32() ==
        AutomationInteropProvider.RootObjectId))
    {
        m.Result = AutomationInteropProvider.ReturnRawElementProvider(
                this.Handle, m.WParam, m.LParam,
                (IRawElementProviderSimple)this);
        return;
    }
    base.WndProc(ref m);
}
```

```
''' <summary>
''' Handles WM_GETOBJECT message; others are passed to base handler.
''' </summary>
''' <param name="m">Windows message.</param>
''' <remarks>
''' This method enables UI Automation to find the control.
''' In this example, the implementation of IRawElementProvider is in the same class
''' as this method.
''' </remarks>
Protected Overrides Sub WndProc(ByRef m As Message)
    Const WM_GETOBJECT As Integer = &H3D

    If m.Msg = WM_GETOBJECT AndAlso m.LParam.ToInt32() = AutomationInteropProvider.RootObjectId Then
        m.Result = AutomationInteropProvider.ReturnRawElementProvider(Me.Handle, m.WParam, m.LParam, 
DirectCast(Me, IRawElementProviderSimple))
        Return
    End If
    MyBase.WndProc(m)

End Sub
```

See also

- UI Automation Providers Overview
- Server-Side UI Automation Provider Implementation

Return Properties from a UI Automation Provider

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains sample code that shows how a UI Automation provider can return properties of an element to client applications.

For any property it does not explicitly support, the provider must return `null` (`Nothing` in Visual Basic). This ensures that UI Automation attempts to obtain the property from another source, such as the host window provider.

Example

```
/// <summary>
/// Gets provider property values.
/// </summary>
/// <param name="propId">Property identifier.</param>
/// <returns>The value of the property.</returns>
object IRawElementProviderSimple.GetPropertyValue(int propId)
{
    if (propId == AutomationElementIdentifiers.NameProperty.Id)
    {
        return "Custom list control";
    }
    else if (propId == AutomationElementIdentifiers.ControlTypeProperty.Id)
    {
        return ControlType.List.Id;
    }
    else if (propId == AutomationElementIdentifiers.IsContentElementProperty.Id)
    {
        return true;
    }
    else if (propId == AutomationElementIdentifiers.IsControlElementProperty.Id)
    {
        return true;
    }
    else
    {
        return null;
    }
}
```

```
''' <summary>
''' Gets provider property values.
''' </summary>
''' <param name="propId">Property identifier.</param>
''' <returns>The value of the property.</returns>
Function GetPropertyValue(ByVal propId As Integer) As Object _
    Implements IRawElementProviderSimple.GetPropertyValue

    If propId = AutomationElementIdentifiers.NameProperty.Id Then
        Return "Custom list control"
    ElseIf propId = AutomationElementIdentifiers.ControlTypeProperty.Id Then
        Return ControlType.List.Id
    ElseIf propId = AutomationElementIdentifiers.IsContentElementProperty.Id Then
        Return True
    ElseIf propId = AutomationElementIdentifiers.IsControlElementProperty.Id Then
        Return True
    Else
        Return Nothing
    End If
End Function 'IRawElementProviderSimple.GetPropertyValue
```

See also

- UI Automation Providers Overview
- Server-Side UI Automation Provider Implementation

Raise Events from a UI Automation Provider

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that shows how to raise an event from a UI Automation provider.

Example

In the following example, a UI Automation event is raised in the implementation of a custom button control. The implementation enables a UI Automation client application to simulate a button click.

To avoid unnecessary processing, the example checks ClientsAreListening to see whether events should be raised.

```
/// <summary>
/// Responds to a button click, regardless of whether it was caused by a mouse or
/// keyboard click or by InvokePattern.Invoke.
/// </summary>
private void OnCustomButtonClicked()
{
    // TODO  Perform program actions invoked by the control.

    // Raise an event.
    if (AutomationInteropProvider.ClientsAreListening)
    {
        AutomationEventArgs args = new AutomationEventArgs(InvokePatternIdentifiers.InvokedEvent);
        AutomationInteropProvider.RaiseAutomationEvent(InvokePatternIdentifiers.InvokedEvent, this, args);
    }
}
```

See also

- UI Automation Providers Overview

Enable Navigation in a UI Automation Fragment Provider

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that shows how to enable navigation in a UI Automation provider for an element that is within a fragment.

Example

The following example code implements Navigate for a list item within a list. The parent element is the list box element, and the sibling elements are other items in the list collection. The method returns `null` (`Nothing` in Visual Basic) for directions that are not valid; in this case, FirstChild and LastChild, because the element has no children.

```
/// <summary>
/// Navigate to adjacent elements in the automation tree.
/// </summary>
/// <param name="direction">Direction to navigate.</param>
/// <returns>The element in that direction, or null.</returns>
/// <remarks>
/// parentControl is the provider for the list box.
/// parentItems is the collection of list item providers.
/// </remarks>
public IRawElementProviderFragment Navigate(NavigateDirection direction)
{
    int myIndex = parentItems.IndexOf(this);
    if (direction == NavigateDirection.Parent)
    {
        return (IRawElementProviderFragment)parentControl;
    }
    else if (direction == NavigateDirection.NextSibling)
    {
        if (myIndex < parentItems.Count - 1)
        {
            return (IRawElementProviderFragment)parentItems[myIndex + 1];
        }
        else
        {
            return null;
        }
    }
    else if (direction == NavigateDirection.PreviousSibling)
    {
        if (myIndex > 0)
        {
            return (IRawElementProviderFragment)parentItems[myIndex - 1];
        }
        else
        {
            return null;
        }
    }
    else
    {
        return null;
    }
}
```

```
''' <summary>
''' Navigate to adjacent elements in the automation tree.
''' </summary>
''' <param name="direction">Direction to navigate.</param>
''' <returns>The element in that direction, or null.</returns>
''' <remarks>
''' parentControl is the provider for the list box.
''' parentItems is the collection of list item providers.
''' </remarks>
Public Function Navigate(ByVal direction As NavigateDirection) As IRawElementProviderFragment _
    Implements IRawElementProviderFragment.Navigate

    Dim myIndex As Integer = parentItems.IndexOf(Me)
    If direction = NavigateDirection.Parent Then
        Return DirectCast(parentControl, IRawElementProviderFragment)
    ElseIf direction = NavigateDirection.NextSibling Then
        If myIndex < parentItems.Count - 1 Then
            Return DirectCast(parentItems((myIndex + 1)), IRawElementProviderFragment)
        Else
            Return Nothing
        End If
    ElseIf direction = NavigateDirection.PreviousSibling Then
        If myIndex > 0 Then
            Return DirectCast(parentItems((myIndex - 1)), IRawElementProviderFragment)
        Else
            Return Nothing
        End If
    Else
        Return Nothing
    End If

End Function 'Navigate
```

See also

- UI Automation Providers Overview
- Server-Side UI Automation Provider Implementation

Support Control Patterns in a UI Automation Provider

11/23/2019 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to implement one or more control patterns on a UI Automation provider so that client applications can manipulate controls and get data from them.

Support Control Patterns

1. Implement the appropriate interfaces for the control patterns that the element should support, such as IInvokeProvider for InvokePattern.
2. Return the object containing your implementation of each control interface in your implementation of IRawElementProviderSimple.GetPatternProvider

Example

The following example shows an implementation of ISelectionProvider for a single-selection custom list box. It returns three properties and gets the currently selected item.

```
#region ISelectionProvider Members

/// <summary>
/// Specifies whether selection of more than one item at a time is supported.
/// </summary>
public bool CanSelectMultiple
{
    get
    {
        return false;
    }
}

/// <summary>
/// Specifies whether the list has to have an item selected at all times.
/// </summary>
public bool IsSelectionRequired
{
    get
    {
        return true;
    }
}

/// <summary>
/// Returns the automation provider for the selected list item.
/// </summary>
/// <returns>The selected item.</returns>
/// <remarks>
/// MyList is an ArrayList collection of providers for items in the list box.
/// SelectedIndex is the index of the selected item.
/// </remarks>
public IRawElementProviderSimple[] GetSelection()
{
    if (SelectedIndex >= 0)
    {
        IRawElementProviderSimple itemProvider = (IRawElementProviderSimple)MyList[SelectedIndex];
        IRawElementProviderSimple[] providers =  { itemProvider };
        return providers;
    }
    else
    {
        return null;
    }
}
#endregion ISelectionProvider Members
```

```
#Region "ISelectionProvider Members"

''' <summary>
''' Specifies whether selection of more than one item at a time is supported.
''' </summary>

Public ReadOnly Property CanSelectMultiple() As Boolean _
    Implements ISelectionProvider.CanSelectMultiple
    Get
        Return False
    End Get
End Property
''' <summary>
''' Specifies whether the list has to have an item selected at all times.
''' </summary>

Public ReadOnly Property IsSelectionRequired() As Boolean _
    Implements ISelectionProvider.IsSelectionRequired
    Get
        Return True
    End Get
End Property

''' <summary>
''' Returns the automation provider for the selected list item.
''' </summary>
''' <returns>The selected item.</returns>
''' <remarks>
''' MyList is an ArrayList collection of providers for items in the list box.
''' SelectedIndex is the index of the selected item.
''' </remarks>
Public Function GetSelection() As IRawElementProviderSimple() _
    Implements ISelectionProvider.GetSelection
    If SelectedIndex >= 0 Then
        Dim itemProvider As IRawElementProviderSimple = DirectCast(MyList(SelectedIndex),
IRawElementProviderSimple)
        Dim providers(1) As IRawElementProviderSimple
        providers(0) = itemProvider
        Return providers
    Else
        Return Nothing
    End If

End Function 'GetSelection
#End Region
Private Members As ISelectionProvider
```

Example

The following example shows an implementation of GetPatternProvider that returns the class implementing ISelectionProvider. Most list box controls would support other patterns as well, but in this example a null reference (`Nothing` in Microsoft Visual Basic .NET) is returned for all other pattern identifiers.

```csharp
/// <summary>
/// Returns the object that supports the specified pattern.
/// </summary>
/// <param name="patternId">ID of the pattern.</param>
/// <returns>Object that implements IInvokeProvider.</returns>
/// <remarks>
/// In this case, the ISelectionProvider interface is implemented in another provider-defined class,
/// ListPattern. However, it could be implemented in the base provider class, in which case the
/// method would simply return "this".
/// </remarks>
object IRawElementProviderSimple.GetPatternProvider(int patternId)
{
    if (patternId == SelectionPatternIdentifiers.Pattern.Id)
    {
        return new ListPattern(myItems, SelectedIndex);
    }
    else
    {
        return null;
    }
}
```

```vb
''' <summary>
''' Returns the object that supports the specified pattern.
''' </summary>
''' <param name="patternId">ID of the pattern.</param>
''' <returns>Object that implements IInvokeProvider.</returns>
''' <remarks>
''' In this case, the ISelectionProvider interface is implemented in another provider-defined class,
''' ListPattern. However, it could be implemented in the base provider class, in which case the
''' method would simply return "this".
''' </remarks>
Function GetPatternProvider(ByVal patternId As Integer) As Object _
    Implements IRawElementProviderSimple.GetPatternProvider

    If patternId = SelectionPatternIdentifiers.Pattern.Id Then
        Return New ListPattern(myItems, SelectedIndex)
    Else
        Return Nothing
    End If

End Function 'IRawElementProviderSimple.GetPatternProvider
```

See also

- UI Automation Providers Overview
- Server-Side UI Automation Provider Implementation

Create a Client-Side UI Automation Provider

11/23/2019 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that shows how to implement a client-side UI Automation provider.

Example

The following example code can be built into a dynamic-link library (DLL) that implements a very simple client-side provider for a console window. The code does not have any useful functionality, but is intended to demonstrate the basic steps in setting up a provider assembly that can be registered by a UI Automation client application.

```
using System;
using System.Windows.Automation;
using System.Windows.Automation.Provider;

namespace ClientSideProviderAssembly
{
    // The assembly must implement a UIAutomationClientSideProviders class,
    // and the namespace must be the same as the name of the DLL, so that
    // UI Automation can find the table of descriptors. In this example,
    // the DLL would be "ClientSideProviderAssembly.dll"

    static class UIAutomationClientSideProviders
    {
        /// <summary>
        /// Implementation of the static ClientSideProviderDescriptionTable field.
        /// In this case, only a single provider is listed in the table.
        /// </summary>
        public static ClientSideProviderDescription[] ClientSideProviderDescriptionTable =
            {
                new ClientSideProviderDescription(
                    // Method that creates the provider object.
                    new ClientSideProviderFactoryCallback(ConsoleProvider.Create),
                    // Class of window that will be served by the provider.
                    "ConsoleWindowClass")
            };
    }

    class ConsoleProvider : IRawElementProviderSimple
    {
        IntPtr providerHwnd;

        public ConsoleProvider(IntPtr hwnd)
        {
            providerHwnd = hwnd;
        }

        internal static IRawElementProviderSimple Create(
            IntPtr hwnd, int idChild, int idObject)
        {
            // This provider doesn't expose children, so never expects
            // nonzero values for idChild.
            if (idChild != 0)
                return null;
            else
                return new ConsoleProvider(hwnd);
        }

        private static IRawElementProviderSimple Create(
            IntPtr hwnd, int idChild)
        {
            // Something is wrong if idChild is not 0.
            if (idChild != 0) return null;
            else return new ConsoleProvider(hwnd);
        }

        #region IRawElementProviderSimple

        // This is a skeleton implementation. The only real functionality
        // at this stage is to return the name of the element and the host
        // window provider, which can supply other properties.

        ProviderOptions IRawElementProviderSimple.ProviderOptions
        {
            get
            {
                return ProviderOptions.ClientSideProvider;
            }
        }

        IRawElementProviderSimple IRawElementProviderSimple.HostRawElementProvider
        {
            get
            {
                return AutomationInteropProvider.HostProviderFromHandle(providerHwnd);
            }
        }

        object IRawElementProviderSimple.GetPropertyValue(int propertyId)
        {
            if (propertyId == AutomationElementIdentifiers.NameProperty.Id)
                return "Custom Console Window";
            else
                return null;
        }

        object IRawElementProviderSimple.GetPatternProvider(int iid)
        {
            return null;
        }
        #endregion IRawElementProviderSimple
    }
}
```

```
Imports System.Windows.Automation
Imports System.Windows.Automation.Provider

Namespace ClientSideProviderAssembly
    ' The assembly must implement a UIAutomationClientSideProviders class,
    ' and the namespace must be the same as the name of the DLL, so that
    ' UI Automation can find the table of descriptors. In this example,
    ' the DLL would be "ClientSideProviderAssembly.dll"

    Friend NotInheritable Class UIAutomationClientSideProviders
        ''' <summary>
        ''' Implementation of the static ClientSideProviderDescriptionTable field.
        ''' In this case, only a single provider is listed in the table.
        ''' </summary>
        Public Shared ClientSideProviderDescriptionTable() As ClientSideProviderDescription = { New
ClientSideProviderDescription(New ClientSideProviderFactoryCallback(AddressOf ConsoleProvider.Create),
"ConsoleWindowClass") }
                    ' Method that creates the provider object.
                    ' Class of window that will be served by the provider.
    End Class

    Friend Class ConsoleProvider
        Implements IRawElementProviderSimple
        Private providerHwnd As IntPtr

        Public Sub New(ByVal hwnd As IntPtr)
            providerHwnd = hwnd
        End Sub

        Friend Shared Function Create(ByVal hwnd As IntPtr, ByVal idChild As Integer, ByVal idObject As
Integer) As IRawElementProviderSimple
            ' This provider doesn't expose children, so never expects
            ' nonzero values for idChild.
            If idChild <> 0 Then
                Return Nothing
            Else
                Return New ConsoleProvider(hwnd)
            End If

        End Function

        Private Shared Function Create(ByVal hwnd As IntPtr, ByVal idChild As Integer) As
IRawElementProviderSimple
            ' Something is wrong if idChild is not 0.
            If idChild <> 0 Then
                Return Nothing
            Else
                Return New ConsoleProvider(hwnd)
            End If
        End Function

        #Region "IRawElementProviderSimple"

        ' This is a skeleton implementation. The only real functionality
        ' at this stage is to return the name of the element and the host
        ' window provider, which can supply other properties.

        Private ReadOnly Property IRawElementProviderSimple_ProviderOptions() As ProviderOptions Implements
IRawElementProviderSimple.ProviderOptions
            Get
                Return ProviderOptions.ClientSideProvider
            End Get
        End Property

        Private ReadOnly Property HostRawElementProvider() As IRawElementProviderSimple Implements
IRawElementProviderSimple.HostRawElementProvider
            Get
                Return AutomationInteropProvider.HostProviderFromHandle(providerHwnd)
            End Get
        End Property

        Private Function GetPropertyValue(ByVal propertyId As Integer) As Object Implements
IRawElementProviderSimple.GetPropertyValue
            If propertyId = AutomationElementIdentifiers.NameProperty.Id Then
                Return "Custom Console Window"
            Else
                Return Nothing
            End If

        End Function

        Private Function GetPatternProvider(ByVal iid As Integer) As Object Implements
        Private Function GetPatternProvider(ByVal iid As Integer) As Object Implements
IRawElementProviderSimple.GetPatternProvider
            Return Nothing
        End Function
        #End Region ' IRawElementProviderSimple
    End Class
End Namespace
```

See also

- UI Automation Providers Overview
- Register a Client-Side Provider Assembly

Implement UI Automation Providers in a Client Application

11/23/2019 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that shows how to implement a client-side UI Automation provider within an application.

This is an uncommon scenario. Most often, a UI Automation client application uses server-side providers, or client-side providers that reside in a DLL.

Example

The following example code implements a simple provider for a console window. The code does not have any useful functionality, but is intended to demonstrate the basic steps in setting up a provider within client code and registering it by using RegisterClientSideProviders.

```
using System;
using System.Windows.Automation;
using System.Windows.Automation.Provider;
using System.Reflection;
using System.Runtime.InteropServices;
using System.IO;

namespace CSClientProviderImplementation
{
    class CSClientSideImplementationProgram
    {
        [DllImport("kernel32.dll")]
        static extern IntPtr GetConsoleWindow();

        static void Main(string[] args)
        {

            ClientSettings.RegisterClientSideProviders(
                UIAutomationClientSideProviders.ClientSideProviderDescriptionTable);

            IntPtr hwnd = GetConsoleWindow();

            // Get an AutomationElement that represents the window.
            AutomationElement elementWindow = AutomationElement.FromHandle(hwnd);
            Console.WriteLine("Found UI Automation client-side provider for:");

            // The name property is furnished by the client-side provider.
            Console.WriteLine(elementWindow.Current.Name);
            Console.WriteLine();

            Console.WriteLine("Press any key to exit.");
            Console.ReadLine();
        }
    }  // CSClientSideImplementationProgram class
```

```
class UIAutomationClientSideProviders
{
    /// <summary>
    /// Implementation of the static ClientSideProviderDescriptionTable field.
    /// In this case,only a single provider is listed in the table.
    /// </summary>
    public static ClientSideProviderDescription[] ClientSideProviderDescriptionTable =
        { new ClientSideProviderDescription(
            // Method that creates the provider object.
            WindowProvider.Create,
            // Class of window that will be served by the provider.
            "ConsoleWindowClass") };
}

class WindowProvider : IRawElementProviderSimple
{
    IntPtr providerHwnd;

    public WindowProvider(IntPtr hwnd)
    {
        providerHwnd = hwnd;
    }

    internal static IRawElementProviderSimple Create(
        IntPtr hwnd, int idChild, int idObject)
    {
        return Create(hwnd, idChild);
    }

    private static IRawElementProviderSimple Create(
        IntPtr hwnd, int idChild)
    {
        // Something is wrong if idChild is not 0.
        if (idChild != 0) return null;
        else return new WindowProvider(hwnd);
    }

    #region IRawElementProviderSimple

    // This is a skeleton implementation. The only real functionality at this stage
    // is to return the name of the element and the host window provider, which can
    // supply other properties.

    ProviderOptions IRawElementProviderSimple.ProviderOptions
    {
        get
        {
            return ProviderOptions.ClientSideProvider;
        }
    }

    IRawElementProviderSimple IRawElementProviderSimple.HostRawElementProvider
    {
        get
        {
            return AutomationInteropProvider.HostProviderFromHandle(providerHwnd);
        }
    }

    object IRawElementProviderSimple.GetPropertyValue(int aid)
    {
        if (AutomationProperty.LookupById(aid) ==
            AutomationElementIdentifiers.NameProperty)
        {
            return "Custom Console Window";
        }
        else
        {
                return null;
            }
        }

        object IRawElementProviderSimple.GetPatternProvider(int iid)
        {
            return null;
        }
        #endregion IRawElementProviderSimple
    }
}
```

```
Imports System.Windows.Automation
Imports System.Windows.Automation.Provider
Imports System.Reflection
Imports System.Runtime.InteropServices
Imports System.IO


Namespace CSClientProviderImplementation
    Friend Class CSClientSideImplementationProgram
        <DllImport("kernel32.dll")>
        Shared Function GetConsoleWindow() As IntPtr
        End Function


        Shared Sub Main(ByVal args() As String)


ClientSettings.RegisterClientSideProviders(UIAutomationClientSideProviders.ClientSideProviderDescriptionTable)

            Dim hwnd As IntPtr = GetConsoleWindow()

            ' Get an AutomationElement that represents the window.
            Dim elementWindow As AutomationElement = AutomationElement.FromHandle(hwnd)
            Console.WriteLine("Found UI Automation client-side provider for:")

            ' The name property is furnished by the client-side provider.
            Console.WriteLine(elementWindow.Current.Name)
            Console.WriteLine()

            Console.WriteLine("Press any key to exit.")
            Console.ReadLine()
        End Sub
    End Class


    Friend Class UIAutomationClientSideProviders
        ''' <summary>
        ''' Implementation of the static ClientSideProviderDescriptionTable field.
        ''' In this case,only a single provider is listed in the table.
        ''' </summary>
        ' Method that creates the provider object.
        ' Class of window that will be served by the provider.
        Public Shared ClientSideProviderDescriptionTable() As ClientSideProviderDescription = {New
ClientSideProviderDescription(AddressOf WindowProvider.Create, "ConsoleWindowClass")}
    End Class

    Friend Class WindowProvider
        Implements IRawElementProviderSimple
        Private providerHwnd As IntPtr

        Public Sub New(ByVal hwnd As IntPtr)
            providerHwnd = hwnd
        End Sub
        Friend Shared Function Create(ByVal hwnd As IntPtr, ByVal idChild As Integer, ByVal idObject As
Integer) As IRawElementProviderSimple
            Return Create(hwnd, idChild)
        End Function

        Private Shared Function Create(ByVal hwnd As IntPtr, ByVal idChild As Integer) As
IRawElementProviderSimple
            ' Something is wrong if idChild is not 0.
            If idChild <> 0 Then
                Return Nothing
            Else
                Return New WindowProvider(hwnd)
            End If
        End Function

#Region "IRawElementProviderSimple"

        ' This is a skeleton implementation. The only real functionality at this stage
        ' is to return the name of the element and the host window provider, which can
        ' supply other properties.

        Private ReadOnly Property IRawElementProviderSimple_ProviderOptions() As ProviderOptions Implements
IRawElementProviderSimple.ProviderOptions
            Get
                Return ProviderOptions.ClientSideProvider
            End Get
        End Property

        Private ReadOnly Property HostRawElementProvider() As IRawElementProviderSimple Implements
IRawElementProviderSimple.HostRawElementProvider
            Get
                Return AutomationInteropProvider.HostProviderFromHandle(providerHwnd)
            End Get
        End Property

        Private Function GetPropertyValue(ByVal aid As Integer) As Object Implements
IRawElementProviderSimple.GetPropertyValue
            If AutomationProperty.LookupById(aid) Is AutomationElementIdentifiers.NameProperty Then
                Return "Custom Console Window"
            Else
                Return Nothing
            End If

        End Function

        Private Function GetPatternProvider(ByVal iid As Integer) As Object Implements
IRawElementProviderSimple.GetPatternProvider
            Return Nothing
        End Function
#End Region ' IRawElementProviderSimple
    End Class
End Namespace
```

See also

- UI Automation Providers Overview
- Register a Client-Side Provider Assembly
- Create a Client-Side UI Automation Provider
- Client-Side UI Automation Provider Implementation

UI Automation Clients for Managed Code

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section contains overviews and how-to topics to help you develop UI Automation clients.

In This Section

UI Automation and Screen Scaling
UI Automation Support for Standard Controls
UI Automation Events for Clients
Caching in UI Automation Clients
UI Automation Properties for Clients
Control Pattern Mapping for UI Automation Clients
UI Automation Control Patterns for Clients
Obtaining UI Automation Elements
UI Automation Threading Issues
How-to Topics

Reference

System.Windows.Automation

UI Automation and Screen Scaling

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Starting with Windows Vista, Windows enables users to change the dots per inch (dpi) setting so that most user interface (UI) elements on the screen appear larger. Although this feature has long been available in Windows, in previous versions the scaling had to be implemented by applications. Starting with Windows Vista, the Desktop Window Manager performs default scaling for all applications that do not handle their own scaling. UI Automation client applications must take this feature into account.

Scaling in Windows Vista

The default dpi setting is 96, which means that 96 pixels occupy a width or height of one notional inch. The exact measure of an "inch" depends on the size and physical resolution of the monitor. For example, on a monitor 12 inches wide, at a horizontal resolution of 1280 pixels, a horizontal line of 96 pixels extends about 9/10 of an inch.

Changing the dpi setting is not the same as changing the screen resolution. With dpi scaling, the number of physical pixels on the screen remains the same. However, scaling is applied to the size and location of UI elements. This scaling can be performed automatically by the Desktop Window Manager (DWM) for the desktop and for applications that do not explicitly ask not to be scaled.

In effect, when the user sets the scale factor to 120 dpi, a vertical or horizontal inch on the screen becomes bigger by 25 percent. All dimensions are scaled accordingly. The offset of an application window from the top and left edges of the screen increases by 25 percent. If application scaling is enabled and the application is not dpi-aware, the size of the window increases in the same proportion, along with the offsets and sizes of all UI elements it contains.

NOTE

By default, the DWM does not perform scaling for non-dpi-aware applications when the user sets the dpi to 120, but does perform it when the dpi is set to a custom value of 144 or higher. However, the user can override the default behavior.

Screen scaling creates new challenges for applications that are concerned in any way with screen coordinates. The screen now contains two coordinate systems: physical and logical. The physical coordinates of a point are the actual offset in pixels from the top left of the origin. The logical coordinates are the offsets as they would be if the pixels themselves were scaled.

Suppose you design a dialog box with a button at coordinates (100, 48). When this dialog box is displayed at the default 96 dpi, the button is located at physical coordinates of (100, 48). At 120 dpi, it is located at physical coordinates of (125, 60). But the logical coordinates are the same at any dpi setting: (100, 48).

Logical coordinates are important, because they make the behavior of the operating system and applications consistent regardless of the dpi setting. For example, Cursor.Position normally returns the logical coordinates. If you move the cursor over an element in a dialog box, the same coordinates are returned regardless of the dpi setting. If you draw a control at (100, 100), it is drawn to those logical coordinates, and will occupy the same relative position at any dpi setting.

Scaling in UI Automation Clients

The UI Automation API does not use logical coordinates. The following methods and properties either return physical coordinates or take them as parameters.

- GetClickablePoint
- TryGetClickablePoint
- ClickablePointProperty
- FromPoint
- BoundingRectangle

By default, a UI Automation client application running in a non-96- dpi environment will not be able to obtain correct results from these methods and properties. For example, because the cursor position is in logical coordinates, the client cannot simply pass these coordinates to FromPoint to obtain the element that is under the cursor. In addition, the application will not be able to correctly place windows outside its client area.

The solution is in two parts.

1. First, make the client application dpi-aware. To do this, call the Win32 function `SetProcessDPIAware` at startup. In managed code, the following declaration makes this function available.

   ```
   [System.Runtime.InteropServices.DllImport("user32.dll")]
   internal static extern bool SetProcessDPIAware();
   ```

   ```
   <System.Runtime.InteropServices.DllImport("user32.dll")> _
   Friend Shared Function SetProcessDPIAware() As Boolean
   End Function
   ```

   This function makes the entire process dpi-aware, meaning that all windows that belong to the process are unscaled. In the Highlighter Sample, for instance, the four windows that make up the highlight rectangle are located at the physical coordinates obtained from UI Automation, not the logical coordinates. If the sample were not dpi-aware, the highlight would be drawn at the logical coordinates on the desktop, which would result in incorrect placement in a non-96- dpi environment.

2. To get cursor coordinates, call the Win32 function `GetPhysicalCursorPos`. The following example shows how to declare and use this function.

```
public struct CursorPoint
{
    public int X;
    public int Y;
}

[System.Runtime.InteropServices.DllImport("user32.dll")]
internal static extern bool GetPhysicalCursorPos(ref CursorPoint lpPoint);

private bool ShowUsage()
{
    CursorPoint cursorPos = new CursorPoint();
    try
    {
        return GetPhysicalCursorPos(ref cursorPos);
    }
    catch (EntryPointNotFoundException) // Not Windows Vista
    {
        return false;
    }
}
```

```
Structure CursorPoint
    Public X As Integer
    Public Y As Integer
End Structure

<System.Runtime.InteropServices.DllImport("user32.dll")> _
Friend Shared Function GetPhysicalCursorPos(ByRef lpPoint As CursorPoint) As Boolean
End Function

Private Function ShowUsage() As Boolean

    Dim cursorPos As New CursorPoint()
    Try
        Return GetPhysicalCursorPos(cursorPos)
    Catch e As EntryPointNotFoundException ' Not Windows Vista
        Return False
    End Try

End Function
```

Caution

Do not use Cursor.Position. The behavior of this property outside client windows in a scaled environment is undefined.

If your application performs direct cross-process communication with non- dpi-aware applications, you may have convert between logical and physical coordinates by using the Win32 functions `PhysicalToLogicalPoint` and `LogicalToPhysicalPoint`.

See also

- Highlighter Sample

UI Automation Support for Standard Controls

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains information about Microsoft UI Automation support for standard controls in applications developed for the WPF, Win32, and Windows Forms frameworks.

Windows Presentation Foundation Controls

All WPF control elements that provide information or support for user interaction have full native support for UI Automation. Other elements, such as panels, are not visible to UI Automation.

Win32 Controls

Most Win32 controls are exposed to Microsoft UI Automation through client-side providers in UIAutomationClientsideProviders.dll. This assembly is automatically registered for use with UI Automation client applications.

Full support is provided only for controls from version 6 of *ComCtrl32.dll*.

The following controls are supported.

CLASS NAME	CONTROL TYPE
Button	Button
Button	RadioButton
Button	Group
Button	CheckBox
Button	Hyperlink
Button	SplitButton
Button	CheckBox
ComboBoxEx32	ComboBox
ComboBox	ComboBox
Edit	Document

CLASS NAME	CONTROL TYPE
Edit	Edit
SysLink	Hyperlink
Static	Text
Static	Image
SysIPAddress32	Custom
SysHeader32	Header/HeaderItem
SysListView32	DataGrid
SysListView32	List
ListBox	List
ListBox	ListItem
#32768	Menu
#32768	MenuItem
msctls_progress32	ProgressBar
RichEdit	Document. See note.
RichEdit20A	Document
RichEdit20W	Document
RichEdit50W	Document
ScrollBar	Slider
msctls_trackbar32	Slider
msctls_updown32	Spinner
msctls_statusbar32	StatusBar
SysTabControl32	Tab
SysTabControl32	TabItem
ToolbarWindow32	ToolBar
ToolbarWindow32	MenuItem

CLASS NAME	CONTROL TYPE
ToolbarWindow32	Button
ToolbarWindow32	CheckBox
ToolbarWindow32	RadioButton
ToolbarWindow32	Separator
tooltips_class32	ToolTip
#32774	ToolTip
ReBarWindow32	Toolbar
SysTreeView32	Tree
SysTreeView32	TreeItem

Note The RichEdit control is supported only for versions shipped with Windows Vista (in RichEd20.dll version 3.1 and later, and MsftEdit.dll version 4.1 and later).

The following controls are not supported.

CLASS NAME	CONTROL TYPE
SysAnimate32	Image
SysPager	Spinner
SysDateTimePick32	Custom
SysMonthCal32	Calendar
MS_WINNOTE	Tooltip
VBBubble	Tooltip
ScrollBar (when used as a standalone control)	Slider
SuperGrid	Custom

Windows Forms Controls

Windows Forms controls are exposed to Microsoft UI Automation through client-side providers in UIAutomationClientsideProviders.dll. This assembly is automatically registered for use with UI Automation client applications.

Typically, Windows Forms controls that are managed wrappers for Win32 common controls are supported by UI Automation. The following controls are supported.

CLASS NAME
Button
CheckBox
CheckedListBox
ColorDialog
ComboBox
FolderBrowser
FontDialog
GroupBox
HscrollBar
ImageList
Label
ListBox
ListView
MainMenu/ContextMenu
MonthCalendar
NotifyIcon
OpenFileDialog
PageSetupDialog
PrintDialog
ProgressBar
RadioButton
RichTextBox
SaveFileDialog
ScrollableControl
SoundPlayer

CLASS NAME
StatusBar
TabControl/TabPage
TextBox
Timer
Toolbar
ToolTip
TrackBar
TreeView
VscrollBar
WebBrowser

The following controls are exposed to Microsoft UI Automation only through their support for Microsoft Active Accessibility. Some functionality may not be available.

CONTROL NAME
BindingSource
DataGrid
DataGridView
DataNavigator
DomainUpDown
ErrorProvider
FlowLayoutPanel
Form
LinkLabel
HelpProvider
MaskedTextBox
MenuStrip/ContextMenuStrip
NumericUpDown

CONTROL NAME
Panel
PictureBox
PrintDocument
PrintPreview-Control
PrintPreview-Dialog
PropertyGrid
UserControl
ToolStrip
TableLayoutPanel
SplitContainer/SplitterPanel
Splitter
RaftingContainer
StatusStrip

See also

- UI Automation Control Types

UI Automation Events for Clients

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic describes how Microsoft UI Automation events are used by UI Automation clients.

UI Automation allows clients to subscribe to events of interest. This capability improves performance by eliminating the need to continually poll all the UI elements in the system to see if any information, structure, or state has changed.

Efficiency is also improved by the ability to listen for events only within a defined scope. For example, a client can listen for focus change events on all UI Automation elements in the tree, or on just one element and its descendants.

NOTE

Do not assume that all possible events are raised by a Microsoft UI Automation provider. For example, not all property changes cause events to be raised by the standard proxy providers for Windows Forms and Win32 controls.

For a broader view of UI Automation events, see UI Automation Events Overview.

Subscribing to Events

Client applications subscribe to events of a particular kind by registering an event handler, using one of the following methods.

METHOD	EVENT TYPE	EVENT ARGUMENTS TYPE	DELEGATE TYPE
AddAutomationFocusChangedEventHandler	Focus change	AutomationFocusChangedEventArgs	AutomationFocusChangedEventHandler
AddAutomationPropertyChangedEventHandler	Property change	AutomationPropertyChangedEventArgs	AutomationPropertyChangedEventHandler
AddStructureChangedEventHandler	Structure change	StructureChangedEventArgs	StructureChangedEventHandler
AddAutomationEventHandler	All other events, identified by an AutomationEvent	AutomationEventArgs or WindowClosedEventArgs	AutomationEventHandler

Before calling the method, you must create a delegate method to handle the event. If you prefer, you can handle different kinds of events in a single method, and pass this method in multiple calls to one of the methods in the table. For example, a single AutomationEventHandler can be set up to handle various events differently according to the EventId.

NOTE

To process window-closed events, cast the argument type that is passed to the event handler as WindowClosedEventArgs. Because the Microsoft UI Automation element for the window is no longer valid, you cannot use the `sender` parameter to retrieve information; use GetRuntimeId instead.

Caution

If your application might receive events from its own UI, do not use your application's UI thread to subscribe to events, or to unsubscribe. Doing so might lead to unpredictable behavior. For more information, see UI Automation Threading Issues.

On shutdown, or when UI Automation events are no longer of interest to the application, UI Automation clients should call one of the following methods.

METHOD	DESCRIPTION
RemoveAutomationEventHandler	Unregisters an event handler that was registered by using AddAutomationEventHandler.
RemoveAutomationFocusChangedEventHandler	Unregisters an event handler that was registered by using AddAutomationFocusChangedEventHandler.
RemoveAutomationPropertyChangedEventHandler	Unregisters an event handler that was registered by using AddAutomationPropertyChangedEventHandler.
RemoveAllEventHandlers	Unregisters all registered event handlers.

For example code, see Subscribe to UI Automation Events.

See also

- Subscribe to UI Automation Events
- UI Automation Events Overview
- UI Automation Properties Overview
- TrackFocus Sample

Caching in UI Automation Clients

3/12/2020 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces caching of UI Automation properties and control patterns.

In UI Automation, caching means pre-fetching of data. The data can then be accessed without further cross-process communication. Caching is typically used by UI Automation client applications to retrieve properties and control patterns in bulk. Information is then retrieved from the cache as needed. The application updates the cache periodically, usually in response to events signifying that something in the user interface (UI) has changed.

The benefits of caching are most noticeable with Windows Presentation Foundation (WPF) controls and custom controls that have server-side UI Automation providers. There is less benefit when accessing client-side providers such as the default providers for Win32 controls.

Caching occurs when the application activates a CacheRequest and then uses any method or property that returns an AutomationElement; for example, FindFirst, FindAll. The methods of the TreeWalker class are an exception; caching is only done if a CacheRequest is specified as a parameter (for example, TreeWalker.GetFirstChild(AutomationElement, CacheRequest).

Caching also occurs when you subscribe to an event while a CacheRequest is active. The AutomationElement passed to your event handler as the source of an event contains the cached properties and patterns specified by the original CacheRequest. Any changes made to the CacheRequest after you subscribe to the event have no effect.

The UI Automation properties and control patterns of an element can be cached.

Options for Caching

The CacheRequest specifies the following options for caching.

Properties to Cache

You can specify properties to cache by calling Add(AutomationProperty) for each property before activating the request.

Control Patterns to Cache

You can specify control patterns to cache by calling Add(AutomationPattern) for each pattern before activating the request. When a pattern is cached, its properties are not automatically cached; you must specify the properties you want cached by using CacheRequest.Add.

Scope and Filtering of Caching

You can specify the elements whose properties and patterns you want to cache by setting the CacheRequest.TreeScope property before activating the request. The scope is relative to the elements that are retrieved while the request is active. For example, if you set only Children, and then retrieve an AutomationElement, the properties and patterns of children of that element are cached, but not those of the element itself. To ensure that caching is done for the retrieved element itself, you must include Element in the TreeScope property. It is not possible to set the scope to Parent or Ancestors. However, a parent element can be cached when a child element is cached; see Retrieving Cached Children and Parents in this topic.

The extent of caching is also affected by the CacheRequest.TreeFilter property. By default, caching is performed only for elements that appear in the control view of the UI Automation tree. However, you can change this property to apply caching to all elements, or only to elements that appear in the content view.

Strength of the Element References

When you retrieve an AutomationElement, by default you have access to all properties and patterns of that element, including those that were not cached. However, for greater efficiency you can specify that the reference to the element refers to cached data only, by setting the AutomationElementMode property of the CacheRequest to None. In this case, you do not have access to any non-cached properties and patterns of retrieved elements. This means that you cannot access any properties through GetCurrentPropertyValue or the `Current` property of AutomationElement or any control pattern; nor can you retrieve a pattern by using GetCurrentPattern or TryGetCurrentPattern. On cached patterns, you can call methods that retrieve array properties, such as SelectionPattern.SelectionPatternInformation.GetSelection, but not any that perform actions on the control, such as InvokePattern.Invoke.

An example of an application that might not need full references to objects is a screen reader, which would prefetch the Name and ControlType properties of elements in a window but would not need the AutomationElement objects themselves.

Activating the CacheRequest

Caching is performed only when AutomationElement objects are retrieved while a CacheRequest is active for the current thread. There are two ways to activate a CacheRequest.

The usual way is to call Activate. This method returns an object that implements IDisposable. The request remains active as long as the IDisposable object exists. The easiest way to control the lifetime of the object is to enclose the call within a `using` (C#) or `Using` (Visual Basic) block. This ensures that the request will be popped from the stack even if an exception is raised.

Another way, which is useful when you wish to nest cache requests, is to call Push. This puts the request on a stack and activates it. The request remains active until it is removed from the stack by Pop. The request becomes temporarily inactive if another request is pushed onto the stack; only the top request on the stack is active.

Retrieving Cached Properties

You can retrieve the cached properties of an element through the following methods and properties.

- GetCachedPropertyValue
- Cached

An exception is raised if the requested property is not in the cache.

Cached, like Current, exposes individual properties as members of a structure. However, you do not need to retrieve this structure; you can access the individual properties directly. For example, the Name property can be obtained from `element.Cached.Name`, where `element` is an AutomationElement.

Retrieving Cached Control Patterns

You can retrieve the cached control patterns of an element through the following methods.

- GetCachedPattern
- TryGetCachedPattern

If the pattern is not in the cache, GetCachedPattern raises an exception, and TryGetCachedPattern returns `false`.

You can retrieve the cached properties of a control pattern by using the `Cached` property of the pattern object. You can also retrieve the current values through the `Current` property, but only if None was not specified when the AutomationElement was retrieved. (Full is the default value, and this permits access to the current values.)

Retrieving Cached Children and Parents

When you retrieve an AutomationElement and request caching for children of that element through the TreeScope property of the request, it is subsequently possible to get the child elements from the CachedChildren property of the element you retrieved.

If Element was included in the scope of the cache request, the root element of the request is subsequently available from the CachedParent property of any of the child elements.

> **NOTE**
>
> You cannot cache parents or ancestors of the root element of the request.

Updating the Cache

The cache is valid only as long as nothing changes in the UI. Your application is responsible for updating the cache, typically in response to events.

If you subscribe to an event while a CacheRequest is active, you obtain an AutomationElement with an updated cache as the source of the event whenever your event-handler delegate is called. You can also update cached information for an element by calling GetUpdatedCache. You can pass in the original CacheRequest to update all information that was previously cached.

Updating the cache does not alter the properties of any existing AutomationElement references.

See also

- UI Automation Events for Clients
- Use Caching in UI Automation
- FetchTimer Sample

UI Automation Properties for Clients

11/23/2019 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This overview introduces you to UI Automation properties as they are exposed to UI Automation client applications.

Properties on AutomationElement objects contain information about user interface (UI) elements, usually controls. The properties of an AutomationElement are generic; that is, not specific to a control type. Many of these properties are exposed in the AutomationElement.AutomationElementInformation structure.

Control patterns also have properties. The properties of control patterns are specific to the pattern. For example, ScrollPattern has properties that enable a client application to discover whether a window is vertically or horizontally scrollable, and what the current view sizes and scroll positions are. Control patterns expose all their properties through a structure; for example, ScrollPattern.ScrollPatternInformation.

UI Automation properties are read-only. To set properties of a control, you must use the methods of the appropriate control pattern. For example, use Scroll to change the position values of a scrolling window.

To improve performance, property values of controls and control patterns can be cached when AutomationElement objects are retrieved. For more information, see Caching in UI Automation Clients.

Property IDs

Property identifiers (IDs) are unique, constant values that are encapsulated in AutomationProperty objects. UI Automation client applications get these IDs from the AutomationElement class or from the appropriate control pattern class, such as ScrollPattern. UI Automation providers get them from AutomationElementIdentifiers or from one of the control pattern identifiers classes, such as ScrollPatternIdentifiers.

The numeric Id of an AutomationProperty is used by providers to identify properties that are being queried for in the IRawElementProviderSimple.GetPropertyValue method. In general, client applications do not need to examine the Id. The ProgrammaticName is used only for debugging and diagnostic purposes.

Property Conditions

The property IDs are used in constructing PropertyCondition objects used to find AutomationElement objects. For example, you might wish to find an AutomationElement that has a certain name, or all controls that are enabled. Each PropertyCondition specifies an AutomationProperty identifier and the value that the property must match.

For more information, see the following reference topics:

- FindFirst
- FindAll

- Condition

Retrieving Properties

Some properties of AutomationElement and all properties of a control pattern class are exposed as nested properties of the `Current` or `Cached` property of the AutomationElement or control pattern object.

In addition, any AutomationElement or control pattern property, including a property that is not available in the Cached or Current structure, can be retrieved by using one of the following methods:

- GetCachedPropertyValue
- GetCurrentPropertyValue

These methods offer slightly better performance as well as access to the full range of properties.

The following code example shows the two ways of retrieving a property on an AutomationElement.

```
// elementList is an AutomationElement.

// The following two calls are equivalent.
string name = elementList.Current.Name;
name = elementList.GetCurrentPropertyValue(AutomationElement.NameProperty) as string;
```

```
' elementList is an AutomationElement.
' The following two calls are equivalent.
Dim name As String = elementList.Current.Name
name = CStr(elementList.GetCurrentPropertyValue(AutomationElement.NameProperty))
```

To retrieve properties of control patterns supported by the AutomationElement, you do not need to retrieve the control pattern object. Simply pass one of the pattern property identifiers to the method.

The following code example shows the two ways of retrieving a property on a control pattern.

```
// elementList is an AutomationElement representing a list box.
// Error-checking is omitted. Assume that elementList is known to support SelectionPattern.

SelectionPattern selectPattern =
    elementList.GetCurrentPattern(SelectionPattern.Pattern) as SelectionPattern;
bool isMultipleSelect = selectPattern.Current.CanSelectMultiple;

// The following call is equivalent to the one above.
isMultipleSelect = (bool)
    elementList.GetCurrentPropertyValue(SelectionPattern.CanSelectMultipleProperty);
```

```
' elementList is an AutomationElement representing a list box.
' Error-checking is omitted. Assume that elementList is known to support SelectionPattern.
Dim selectPattern As SelectionPattern = _
    DirectCast(elementList.GetCurrentPattern(SelectionPattern.Pattern), SelectionPattern)
Dim isMultipleSelect As Boolean = selectPattern.Current.CanSelectMultiple

' The following call is equivalent to the one above.
isMultipleSelect =
CBool(elementList.GetCurrentPropertyValue(SelectionPattern.CanSelectMultipleProperty))
```

The `Get` methods return an Object. The application must cast the returned object to the proper type before using the value.

Default Property Values

If a UI Automation provider does not implement a property, the UI Automation system is able to supply a default value. For example, if the provider for a control does not support the property identified by HelpTextProperty, UI Automation returns an empty string. Similarly, if the provider does not support the property identified by IsDockPatternAvailableProperty, UI Automation returns `false`.

You can change this behavior by using the AutomationElement.GetCachedPropertyValue and AutomationElement.GetCurrentPropertyValue method overloads. When you specify `true` as the second parameter, UI Automation does not return a default value, but instead returns the special value NotSupported.

The following example code attempts to retrieve a property from an element, and if the property is not supported, an application-defined value is used instead.

```
// elementList is an AutomationElement.
object help = elementList.GetCurrentPropertyValue(AutomationElement.HelpTextProperty, true);
if (help == AutomationElement.NotSupported)
{
    help = "No help available";
}
string helpText = (string)help;
```

```
' elementList is an AutomationElement.
Dim help As Object = elementList.GetCurrentPropertyValue(AutomationElement.HelpTextProperty, True)
If help Is AutomationElement.NotSupported Then
    help = "No help available"
End If
Dim helpText As String = CStr(help)
```

To discover what properties are supported by an element, use GetSupportedProperties. This returns an array of AutomationProperty identifiers.

Property-changed Events

When a property value on an AutomationElement or control pattern changes, an event is raised. An application can subscribe to such events by calling AddAutomationPropertyChangedEventHandler, supplying an array of AutomationProperty identifiers as the last parameter in order to specify the properties of interest.

In the AutomationPropertyChangedEventHandler, you can identify the property that has changed by checking the Property member of the event arguments. The arguments also contain the old and new values of the UI Automation property that has changed. These values are of type Object and must be cast to the correct type before being used.

Additional AutomationElement Properties

In addition to the Current and Cached property structures, AutomationElement has the following properties, which are retrieved through simple property accessors.

PROPERTY	DESCRIPTION
CachedChildren	A collection of child AutomationElement objects that are in the cache.

PROPERTY	DESCRIPTION
CachedParent	An AutomationElement parent object that is in the cache.
FocusedElement	(Static property) The AutomationElement that has the input focus.
RootElement	(Static property) The root AutomationElement.

See also

- Caching in UI Automation Clients
- Server-Side UI Automation Provider Implementation
- Subscribe to UI Automation Events

Control Pattern Mapping for UI Automation Clients

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic lists control types and their associated control patterns.

The following table organizes the control patterns into the following categories:

- Supported. The control must support this control pattern.
- Conditional support. The control may support this control pattern depending on the state of the control.
- Not supported. The control does not support this control pattern; custom controls may support this control pattern.

NOTE

Some controls have conditional support for several control patterns depending on the functionality of the control. For example, the menu item control has conditional support for the InvokePattern, ExpandCollapsePattern, TogglePattern, or SelectionItemPattern control pattern, depending on its function in the menu control.

UI Automation Control Patterns for Clients

CONTROL TYPE	SUPPORTED	CONDITIONAL SUPPORT	NOT SUPPORTED
Button	None	Invoke, Toggle, Expand Collapse	None
Calendar	Grid, Table	Selection, Scroll	Value
Check Box	Toggle	None	None
Combo Box	Expand Collapse	Selection, Value	Scroll
Data Grid	Grid	Scroll, Selection, Table	None
Data Item	Selection Item	Expand Collapse, Grid Item, Scroll Item, Table, Toggle, Value	None
Document	Text	Scroll, Value	None
Edit	None	Text, Range Value, Value	None
Group	None	Expand Collapse	None

CONTROL TYPE	SUPPORTED	CONDITIONAL SUPPORT	NOT SUPPORTED
Header	None	Transform	None
Header Item	None	Transform, Invoke	None
Hyperlink	Invoke	Value	None
Image	None	Grid Item, Table Item	Invoke, Selection Item
List	None	Grid, Multiple View, Scroll, Selection	Table
List Item	Selection Item	Expand Collapse, Grid Item, Invoke, Scroll Item, Toggle, Value	None
Menu	None	None	None
Menu Bar	None	Expand Collapse, Dock, Transform	None
Menu Item	None	Expand Collapse, Invoke, Selection Item, Toggle	None
Pane	None	Dock. Scroll, Transform	Window
Progress Bar	None	Range Value, Value	None
Radio Button	Selection Item	None	Toggle
Scroll Bar	None	Range Value	Scroll
Separator	None	None	None
Slider	None	Range Value, Selection, Value	None
Spinner	None	Range Value, Selection, Value	None
Split Button	Invoke, Expand Collapse	None	None
Status Bar	None	Grid	None
Tab	Selection	Scroll	None
Tab Item	Selection Item	None	Invoke
Table	Grid, Grid Item, Table, Table Item	None	None
Text	None	Grid Item, Table Item, Text	Value

CONTROL TYPE	SUPPORTED	CONDITIONAL SUPPORT	NOT SUPPORTED
Thumb	Transform	None	None
Title Bar	None	None	None
Tool Bar	None	Dock, Expand Collapse, Transform	None
Tool Tip	None	Text, Window	None
Tree	None	Scroll, Selection	None
Tree Item	Expand Collapse	Invoke, Scroll Item, Selection Item, Toggle	None
Window	Transform, Window	Dock	None

NOTE

If a control type has no supported control patterns listed but has one or more conditionally-supported control patterns, then one of those conditional control patterns will be supported at all times.

See also

- UI Automation Overview

UI Automation Control Patterns for Clients

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This overview introduces control patterns for UI Automation clients. It includes information on how a UI Automation client can use control patterns to access information about the user interface (UI).

Control patterns provide a way to categorize and expose a control's functionality independent of the control type or the appearance of the control. UI Automation clients can examine an AutomationElement to determine which control patterns are supported and be assured of the behavior of the control.

For a complete list of control patterns, see UI Automation Control Patterns Overview.

Getting Control Patterns

Clients retrieve a control pattern from an AutomationElement by calling either AutomationElement.GetCachedPattern or AutomationElement.GetCurrentPattern.

Clients can use the GetSupportedPatterns method or an individual `IsPatternAvailable` property (for example, IsTextPatternAvailableProperty) to determine if a pattern or group of patterns is supported on the AutomationElement. However, it is more efficient to attempt to get the control pattern and test for a `null` reference than to check the supported properties and retrieve the control pattern since it results in fewer cross-process calls.

The following example demonstrates how to get a TextPattern control pattern from an AutomationElement.

```
// Specify the control type we're looking for, in this case 'Document'
PropertyCondition cond = new PropertyCondition(AutomationElement.ControlTypeProperty,
ControlType.Document);

// target --> The root AutomationElement.
AutomationElement textProvider = target.FindFirst(TreeScope.Descendants, cond);

targetTextPattern = textProvider.GetCurrentPattern(TextPattern.Pattern) as TextPattern;

if (targetTextPattern == null)
{
    Console.WriteLine("Root element does not contain a descendant that supports TextPattern.");
    return;
}
```

Retrieving Properties on Control Patterns

Clients can retrieve the property values on control patterns by calling either AutomationElement.GetCachedPropertyValue or AutomationElement.GetCurrentPropertyValue and casting the object returned to an appropriate type. For more information on UI Automation properties, see UI Automation Properties for Clients.

In addition to the `GetPropertyValue` methods, property values can be retrieved through the common language runtime (CLR) accessors to access the UI Automation properties on a pattern.

Controls with Variable Patterns

Some control types support different patterns depending on their state or the manner in which the control is being used. Examples of controls that can have variable patterns are list views (thumbnails, tiles, icons, list, details), Microsoft Excel Charts (Pie, Line, Bar, Cell Value with a formula), Microsoft Word's document area (Normal, Web Layout, Outline, Print Layout, Print Preview), and Microsoft Windows Media Player skins.

Controls implementing custom control types can have any set of control patterns that are needed to represent their functionality.

See also

- UI Automation Control Patterns
- UI Automation Text Pattern
- Invoke a Control Using UI Automation
- Get the Toggle State of a Check Box Using UI Automation
- Control Pattern Mapping for UI Automation Clients
- TextPattern Insert Text Sample
- TextPattern Search and Selection Sample
- InvokePattern, ExpandCollapsePattern, and TogglePattern Sample

Obtaining UI Automation Elements

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic describes the various ways of obtaining AutomationElement objects for user interface (UI) elements.

Caution

If your client application might attempt to find elements in its own user interface, you must make all UI Automation calls on a separate thread. For more information, see UI Automation Threading Issues.

Root Element

All searches for AutomationElement objects must have a starting-place. This can be any element, including the desktop, an application window, or a control.

The root element for the desktop, from which all elements are descended, is obtained from the static AutomationElement.RootElement property.

Caution

In general, you should try to obtain only direct children of the RootElement. A search for descendants may iterate through hundreds or even thousands of elements, possibly resulting in a stack overflow. If you are attempting to obtain a specific element at a lower level, you should start your search from the application window or from a container at a lower level.

Conditions

For most techniques you can use to retrieve UI Automation elements, you must specify a Condition, which is a set of criteria defining what elements you want to retrieve.

The simplest condition is TrueCondition, a predefined object specifying that all elements within the search scope are to be returned. FalseCondition, the converse of TrueCondition, is less useful, as it would prevent any elements from being found.

Three other predefined conditions can be used alone or in combination with other conditions: ContentViewCondition, ControlViewCondition, and RawViewCondition. RawViewCondition, used by itself, is equivalent to TrueCondition, because it does not filter elements by their IsControlElement or IsContentElement properties.

Other conditions are built up from one or more PropertyCondition objects, each of which specifies a property value. For example, a PropertyCondition might specify that the element is enabled, or that it supports a certain control pattern.

Conditions can be combined using Boolean logic by constructing objects of types AndCondition, OrCondition, and NotCondition.

Search Scope

Searches done by using FindFirst or FindAll must have a scope as well as a starting-place.

The scope defines the space around the starting-place that is to be searched. This might include the element itself, its siblings, its parent, its ancestors, its immediate children, and its descendants.

The scope of a search is defined by a bitwise combination of values from the TreeScope enumeration.

Finding a Known Element

To find a known element, identified by its Name, AutomationId, or some other property or combination of properties, it is easiest to use the FindFirst method. If the element sought is an application window, the starting-point of the search can be the RootElement.

This way of finding UI Automation elements is most useful in automated testing scenarios.

Finding Elements in a Subtree

To find all elements meeting specific criteria that are related to a known element, you can use FindAll. For example, you could use this method to retrieve list items or menu items from a list or menu, or to identify all controls in a dialog box.

Walking a Subtree

If you have no prior knowledge of the applications that your client may be used with, you can construct a subtree of all elements of interest by using the TreeWalker class. Your application might do this in response to a focus-changed event; that is, when an application or control receives input focus, the UI Automation client examines children and perhaps all descendants of the focused element.

Another way in which TreeWalker can be used is to identify the ancestors of an element. For example, by walking up the tree you can identify the parent window of a control.

You can use TreeWalker either by creating an object of the class (defining the elements of interest by passing a Condition), or by using one of the following predefined objects that are defined as fields of TreeWalker.

ContentViewWalker	Finds only elements whose IsContentElement property is `true`.
ControlViewWalker	Finds only elements whose IsControlElement property is `true`.
RawViewWalker	Finds all elements.

After you have obtained a TreeWalker, using it is straightforward. Simply call the `Get` methods to navigate among elements of the subtree.

The Normalize method can be used for navigating to an element in the subtree from another element that is not part of the view. For example, suppose you have created a view of a subtree by using ContentViewWalker. Your application then receives notification that a scroll bar has received the input focus. Because a scroll bar is not a content element, it is not present in your view of the subtree. However, you can pass the AutomationElement representing the scroll bar to Normalize and retrieve the nearest ancestor that is in the content view.

Other Ways to Retrieve an Element

In addition to searches and navigation, you can retrieve an AutomationElement in the following ways.

From an Event

When your application receives a UI Automation event, the source object passed to your event handler is an AutomationElement. For example, if you have subscribed to focus-changed events, the source passed to your AutomationFocusChangedEventHandler is the element that received the focus.

For more information, see Subscribe to UI Automation Events.

From a Point

If you have screen coordinates (for example, a cursor position), you can retrieve an AutomationElement by using the static FromPoint method.

From a Window Handle

To retrieve an AutomationElement from an HWND, use the static FromHandle method.

From the Focused Control

You can retrieve an AutomationElement that represents the focused control from the static FocusedElement property.

See also

- Find a UI Automation Element Based on a Property Condition
- Navigate Among UI Automation Elements with TreeWalker
- UI Automation Tree Overview

UI Automation Threading Issues

12/4/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Because of the way Microsoft UI Automation uses Windows messages, conflicts can occur when a client application attempts to interact with its own UI on the UI thread. These conflicts can lead to very slow performance or even cause the application to stop responding.

If your client application is intended to interact with all elements on the desktop, including its own UI, you should make all UI Automation calls on a separate thread. This includes locating elements (for example, by using TreeWalker or the FindAll method) and using control patterns.

It is safe to make UI Automation calls within a UI Automation event handler, because the event handler is always called on a non-UI thread. However, when subscribing to events that may originate from your client application's UI, you must make the call to AddAutomationEventHandler, or a related method, on a non-UI thread. Remove event handlers on the same thread.

UI Automation Clients for Managed Code How-to Topics

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section provides detailed information about implementing features of Microsoft UI Automation in a client application.

In This Section

Find a UI Automation Element Based on a Property Condition
Navigate Among UI Automation Elements with TreeWalker
Find a UI Automation Element for a List Item
Get UI Automation Element Properties
Use Caching in UI Automation
Subscribe to UI Automation Events
Register a Client-Side Provider Assembly
Use the AutomationID Property

Find a UI Automation Element Based on a Property Condition

11/23/2019 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that shows how to locate an element within the UI Automation tree based on a specific property or properties.

Example

In the following example, a set of property conditions are specified that identify a certain element (or elements) of interest in the AutomationElement tree. A search for all matching elements is then performed with the FindAll method that incorporates a series of AndCondition boolean operations to limit the number of matching elements.

NOTE

When searching from the RootElement, you should try to obtain only direct children. A search for descendants might iterate through hundreds or even thousands of elements, possibly resulting in a stack overflow. If you are attempting to obtain a specific element at a lower level, you should start your search from the application window or from a container at a lower level.

```
///--------------------------------------------------------------------
/// <summary>
/// Walks the UI Automation tree of the target and reports the control
/// type of each element it finds in the control view to the client.
/// </summary>
/// <param name="targetTreeViewElement">
/// The root of the search on this iteration.
/// </param>
/// <param name="elementIndex">
/// The TreeView index for this iteration.
/// </param>
/// <remarks>
/// This is a recursive function that maps out the structure of the
/// subtree of the target beginning at the AutomationElement passed in
/// as the rootElement on the first call. This could be, for example,
/// an application window.
/// CAUTION: Do not pass in AutomationElement.RootElement. Attempting
/// to map out the entire subtree of the desktop could take a very
/// long time and even lead to a stack overflow.
/// </remarks>
///--------------------------------------------------------------------
private void FindTreeViewDescendants(
    AutomationElement targetTreeViewElement, int treeviewIndex)
{
    if (targetTreeViewElement == null)
        return;

    AutomationElement elementNode =
        TreeWalker.ControlViewWalker.GetFirstChild(targetTreeViewElement);

    while (elementNode != null)
    {
        Label elementInfo = new Label();
        elementInfo.Margin = new Thickness(0);
        clientTreeViews[treeviewIndex].Children.Add(elementInfo);

        // Compile information about the control.
        elementInfoCompile = new StringBuilder();
        string controlName =
            (elementNode.Current.Name == "") ?
            "Unnamed control" : elementNode.Current.Name;
        string autoIdName =
            (elementNode.Current.AutomationId == "") ?
            "No AutomationID" : elementNode.Current.AutomationId;

        elementInfoCompile.Append(controlName)
            .Append(" (")
            .Append(elementNode.Current.ControlType.LocalizedControlType)
            .Append(" - ")
            .Append(autoIdName)
            .Append(")");

        // Test for the control patterns of interest for this sample.
        object objPattern;
        ExpandCollapsePattern expcolPattern;
        if (true == elementNode.TryGetCurrentPattern(ExpandCollapsePattern.Pattern, out objPattern))
        {
            expcolPattern = objPattern as ExpandCollapsePattern;
            if (expcolPattern.Current.ExpandCollapseState != ExpandCollapseState.LeafNode)
            {
                Button expcolButton = new Button();
                expcolButton.Margin = new Thickness(0, 0, 0, 5);
                expcolButton.Height = 20;
                expcolButton.Width = 100;
                expcolButton.Content = "ExpandCollapse";
                expcolButton.Tag = expcolPattern;
                expcolButton.Click +=
                    new RoutedEventHandler(ExpandCollapse_Click);
                clientTreeViews[treeviewIndex].Children.Add(expcolButton);
            }
        }
        TogglePattern togPattern;
        if (true == elementNode.TryGetCurrentPattern(TogglePattern.Pattern, out objPattern))
        {
            togPattern = objPattern as TogglePattern;
            Button togButton = new Button();
            togButton.Margin = new Thickness(0, 0, 0, 5);
            togButton.Height = 20;
            togButton.Width = 100;
            togButton.Content = "Toggle";
            togButton.Tag = togPattern;
            togButton.Click += new RoutedEventHandler(Toggle_Click);
            clientTreeViews[treeviewIndex].Children.Add(togButton);
        }
        InvokePattern invPattern;
        if (true == elementNode.TryGetCurrentPattern(InvokePattern.Pattern, out objPattern))
        {
            invPattern = objPattern as InvokePattern;
            Button invButton = new Button();
            invButton.Margin = new Thickness(0);
            invButton.Height = 20;
            invButton.Width = 100;
            invButton.Content = "Invoke";
            invButton.Tag = invPattern;
            invButton.Click += new RoutedEventHandler(Invoke_Click);
            clientTreeViews[treeviewIndex].Children.Add(invButton);
        // Display compiled information about the control.
        elementInfo.Content = elementInfoCompile;
        Separator sep = new Separator();
        clientTreeViews[treeviewIndex].Children.Add(sep);

        // Iterate to next element.
        // elementNode - Current element.
        // treeviewIndex - Index of parent TreeView.
        FindTreeViewDescendants(elementNode, treeviewIndex);
        elementNode =
            TreeWalker.ControlViewWalker.GetNextSibling(elementNode);
    }
}
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Walks the UI Automation tree of the target and reports the control
''' type of each element it finds in the control view to the client.
''' </summary>
''' <param name="targetTreeViewElement">
''' The root of the search on this iteration.
''' </param>
''' <param name="treeviewIndex">
''' The TreeView index for this iteration.
''' </param>
''' <remarks>
''' This is a recursive function that maps out the structure of the
''' subtree beginning at the AutomationElement passed in as
''' rootElement on the first call. This could be, for example,
''' an application window.
''' CAUTION: Do not pass in AutomationElement.RootElement. Attempting
''' to map out the entire subtree of the desktop could take a very
''' long time and even lead to a stack overflow.
''' </remarks>
'''--------------------------------------------------------------------
Private Sub FindTreeViewDescendants( _
ByVal targetTreeViewElement As AutomationElement, _
ByVal treeviewIndex As Integer)
    If (IsNothing(targetTreeViewElement)) Then
        Return
    End If

    Dim elementNode As AutomationElement = _
    TreeWalker.ControlViewWalker.GetFirstChild(targetTreeViewElement)

    While Not (elementNode Is Nothing)
        Dim elementInfo As New Label()
        elementInfo.Margin = New Thickness(0)
        clientTreeViews(treeviewIndex).Children.Add(elementInfo)

        ' Compile information about the control.
        elementInfoCompile = New StringBuilder()
        Dim controlName As String
        If (elementNode.Current.Name = "") Then
            controlName = "Unnamed control"
        Else
            controlName = elementNode.Current.Name
        End If
        Dim autoIdName As String
        If (elementNode.Current.AutomationId = "") Then
            autoIdName = "No AutomationID"
        Else
            autoIdName = elementNode.Current.AutomationId
        End If


        elementInfoCompile.Append(controlName).Append(" (")
        elementInfoCompile.Append(controlName).Append(" (") _
        .Append(elementNode.Current.ControlType.LocalizedControlType) _
        .Append(" - ").Append(autoIdName).Append(")")

        ' Test for the control patterns of interest for this sample.
        Dim objPattern As Object = Nothing
        Dim expcolPattern As ExpandCollapsePattern
        If True = elementNode.TryGetCurrentPattern(ExpandCollapsePattern.Pattern, objPattern) Then
            expcolPattern = DirectCast(objPattern, ExpandCollapsePattern)
            If expcolPattern.Current.ExpandCollapseState <> ExpandCollapseState.LeafNode Then
                Dim expcolButton As New Button()
                expcolButton.Margin = New Thickness(0, 0, 0, 5)
                expcolButton.Height = 20
                expcolButton.Width = 100
                expcolButton.Content = "ExpandCollapse"
                expcolButton.Tag = expcolPattern
                AddHandler expcolButton.Click, AddressOf ExpandCollapse_Click
                clientTreeViews(treeviewIndex).Children.Add(expcolButton)
            End If
        End If
        Dim togPattern As TogglePattern
        If True = elementNode.TryGetCurrentPattern(TogglePattern.Pattern, objPattern) Then
            togPattern = DirectCast(objPattern, TogglePattern)
            Dim togButton As New Button()
            togButton.Margin = New Thickness(0, 0, 0, 5)
            togButton.Height = 20
            togButton.Width = 100
            togButton.Content = "Toggle"
            togButton.Tag = togPattern
            AddHandler togButton.Click, AddressOf Toggle_Click
            clientTreeViews(treeviewIndex).Children.Add(togButton)
        End If
        Dim invPattern As InvokePattern
        If True = elementNode.TryGetCurrentPattern(InvokePattern.Pattern, objPattern) Then
            invPattern = DirectCast(objPattern, InvokePattern)
            Dim invButton As New Button()
            invButton.Margin = New Thickness(0)
            invButton.Height = 20
            invButton.Width = 100
            invButton.Content = "Invoke"
            invButton.Tag = invPattern
            AddHandler invButton.Click, AddressOf Invoke_Click
            clientTreeViews(treeviewIndex).Children.Add(invButton)
        End If
        ' Display compiled information about the control.
        elementInfo.Content = elementInfoCompile
        Dim sep As New Separator()
        clientTreeViews(treeviewIndex).Children.Add(sep)

        ' Iterate to next element.
        ' elementNode - Current element.
        ' treeviewIndex - Index of parent TreeView.
        FindTreeViewDescendants(elementNode, treeviewIndex)
        elementNode = TreeWalker.ControlViewWalker.GetNextSibling(elementNode)
    End While

End Sub
```

See also

- InvokePattern and ExpandCollapsePattern Menu Item Sample
- Obtaining UI Automation Elements
- Use the AutomationID Property

Navigate Among UI Automation Elements with TreeWalker

11/23/2019 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that shows how to navigate among Microsoft UI Automation elements by using the TreeWalker class.

Example

The following example uses GetParent to walk up the Microsoft UI Automation tree until it finds the root element, or desktop. The element just below that is the parent window of the specified element.

```
/// <summary>
/// Retrieves the top-level window that contains the specified UI Automation element.
/// </summary>
/// <param name="element">The contained element.</param>
/// <returns>The containing top-level window element.</returns>
private AutomationElement GetTopLevelWindow(AutomationElement element)
{
    TreeWalker walker = TreeWalker.ControlViewWalker;
    AutomationElement elementParent;
    AutomationElement node = element;
    if (node == elementRoot) return node;
    do
    {
        elementParent = walker.GetParent(node);
        if (elementParent == AutomationElement.RootElement) break;
        node = elementParent;
    }
    while (true);
    return node;
}
```

```
    ''' <summary>
    ''' Retrieves the top-level window that contains the specified UI Automation element.
    ''' </summary>
    ''' <param name="element">The contained element.</param>
    ''' <returns>The containing top-level window element.</returns>
    Private Function GetTopLevelWindow(ByVal element As AutomationElement) As AutomationElement
        Dim walker As TreeWalker = TreeWalker.ControlViewWalker
        Dim elementParent As AutomationElement
        Dim node As AutomationElement = element
        If node = elementRoot Then
            Return node
        End If
        Do
            elementParent = walker.GetParent(node)
            If elementParent = AutomationElement.RootElement Then
                Exit Do
            End If
            node = elementParent
        Loop While True
        Return node

    End Function 'GetTopLevelWindow
End Class
```

Example

The following example uses GetFirstChild and GetNextSibling to create a TreeView that shows an entire subtree of Microsoft UI Automation elements that are in the control view and that are enabled.

```
/// <summary>
/// Walks the UI Automation tree and adds the control type of each enabled control
/// element it finds to a TreeView.
/// </summary>
/// <param name="rootElement">The root of the search on this iteration.</param>
/// <param name="treeNode">The node in the TreeView for this iteration.</param>
/// <remarks>
/// This is a recursive function that maps out the structure of the subtree beginning at the
/// UI Automation element passed in as rootElement on the first call. This could be, for example,
/// an application window.
/// CAUTION: Do not pass in AutomationElement.RootElement. Attempting to map out the entire subtree of
/// the desktop could take a very long time and even lead to a stack overflow.
/// </remarks>
private void WalkEnabledElements(AutomationElement rootElement, TreeNode treeNode)
{
    Condition condition1 = new PropertyCondition(AutomationElement.IsControlElementProperty, true);
    Condition condition2 = new PropertyCondition(AutomationElement.IsEnabledProperty, true);
    TreeWalker walker = new TreeWalker(new AndCondition(condition1, condition2));
    AutomationElement elementNode = walker.GetFirstChild(rootElement);
    while (elementNode != null)
    {
        TreeNode childTreeNode = treeNode.Nodes.Add(elementNode.Current.ControlType.LocalizedControlType);
        WalkEnabledElements(elementNode, childTreeNode);
        elementNode = walker.GetNextSibling(elementNode);
    }
}
```

```
''' <summary>
''' Walks the UI Automation tree and adds the control type of each enabled control
''' element it finds to a TreeView.
''' </summary>
''' <param name="rootElement">The root of the search on this iteration.</param>
''' <param name="treeNode">The node in the TreeView for this iteration.</param>
''' <remarks>
''' This is a recursive function that maps out the structure of the subtree beginning at the
''' UI Automation element passed in as rootElement on the first call. This could be, for example,
''' an application window.
''' CAUTION: Do not pass in AutomationElement.RootElement. Attempting to map out the entire subtree of
''' the desktop could take a very long time and even lead to a stack overflow.
''' </remarks>
Private Sub WalkEnabledElements(ByVal rootElement As AutomationElement, ByVal treeNode As TreeNode)
    Dim condition1 As New PropertyCondition(AutomationElement.IsControlElementProperty, True)
    Dim condition2 As New PropertyCondition(AutomationElement.IsEnabledProperty, True)
    Dim walker As New TreeWalker(New AndCondition(condition1, condition2))
    Dim elementNode As AutomationElement = walker.GetFirstChild(rootElement)
    While (elementNode IsNot Nothing)
        Dim childTreeNode As TreeNode =
treeNode.Nodes.Add(elementNode.Current.ControlType.LocalizedControlType)
        WalkEnabledElements(elementNode, childTreeNode)
        elementNode = walker.GetNextSibling(elementNode)
    End While

End Sub
```

See also

- Obtaining UI Automation Elements

Find a UI Automation Element for a List Item

1/8/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to retrieve an AutomationElement for an item within a list when the index of the item is known.

Example

The following example shows two ways of retrieving a specified item from a list, one using TreeWalker and the other using FindAll.

The first technique tends to be faster for Win32 controls, but the second is faster for Windows Presentation Foundation (WPF) controls.

```
/// <summary>
/// Retrieves an element in a list, using TreeWalker.
/// </summary>
/// <param name="parent">The list element.</param>
/// <param name="index">The index of the element to find.</param>
/// <returns>The list item.</returns>
AutomationElement FindChildAt(AutomationElement parent, int index)
{
    if (index < 0)
    {
        throw new ArgumentOutOfRangeException();
    }
    TreeWalker walker = TreeWalker.ControlViewWalker;
    AutomationElement child = walker.GetFirstChild(parent);
    for (int x = 1; x <= index; x++)
    {
        child = walker.GetNextSibling(child);
        if (child == null)
        {
            throw new ArgumentOutOfRangeException();
        }
    }
    return child;
}

/// <summary>
/// Retrieves an element in a list, using FindAll.
/// </summary>
/// <param name="parent">The list element.</param>
/// <param name="index">The index of the element to find.</param>
/// <returns>The list item.</returns>
AutomationElement FindChildAtB(AutomationElement parent, int index)
{
    Condition findCondition = new PropertyCondition(AutomationElement.IsControlElementProperty, true);
    AutomationElementCollection found = parent.FindAll(TreeScope.Children, findCondition);
    if ((index < 0) || (index >= found.Count))
    {
        throw new ArgumentOutOfRangeException();
    }
    return found[index];
}
```

```
''' <summary>
''' Retrieves an element in a list, using TreeWalker.
''' </summary>
''' <param name="parent">The list element.</param>
''' <param name="index">The index of the element to find.</param>
''' <returns>The list item.</returns>
Function FindChildAt(ByVal parent As AutomationElement, ByVal index As Integer) As AutomationElement

    If (index < 0) Then
        Throw New ArgumentOutOfRangeException()
    End If
    Dim walker As TreeWalker = TreeWalker.ControlViewWalker
    Dim child As AutomationElement = walker.GetFirstChild(parent)
    For x As Integer = 1 To (index - 1)
        child = walker.GetNextSibling(child)
        If child = Nothing Then

            Throw New ArgumentOutOfRangeException()
        End If
    Next x
    Return child
End Function

''' <summary>
''' Retrieves an element in a list, using FindAll.
''' </summary>
''' <param name="parent">The list element.</param>
''' <param name="index">The index of the element to find.</param>
''' <returns>The list item.</returns>
Function FindChildAtB(ByVal parent As AutomationElement, ByVal index As Integer) As AutomationElement
    Dim findCondition As Condition = _
        New PropertyCondition(AutomationElement.IsControlElementProperty, True)
    Dim found As AutomationElementCollection = parent.FindAll(TreeScope.Children, findCondition)
    If (index < 0) Or (index >= found.Count) Then
        Throw New ArgumentOutOfRangeException()
    End If
    Return found(index)
End Function
```

See also

- Obtaining UI Automation Elements

Get UI Automation Element Properties

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to retrieve properties of a UI Automation element.

Get a Current Property Value

1. Obtain the AutomationElement whose property you wish to get.
2. Call GetCurrentPropertyValue, or retrieve the Current property structure and get the value from one of its members.

Get a Cached Property Value

1. Obtain the AutomationElement whose property you wish to get. The property must have been specified in the CacheRequest.
2. Call GetCachedPropertyValue, or retrieve the Cached property structure and get the value from one of its members.

Example

The following example shows various ways to retrieve current properties of an AutomationElement.

```
void PropertyCallsExample(AutomationElement elementList)
{
    // The following two calls are equivalent.
    string name = elementList.Current.Name;
    name = elementList.GetCurrentPropertyValue(AutomationElement.NameProperty) as string;

    // The following shows how to ignore the default property, which
    //  would probably be an empty string if the property is not supported.
    //  Passing "false" as the second parameter is equivalent to using the overload
    //  that does not have this parameter.
    object help = elementList.GetCurrentPropertyValue(AutomationElement.HelpTextProperty, true);
    if (help == AutomationElement.NotSupported)
    {
        help = "No help available";
    }
    string helpText = (string)help;
}
```

```vb
Sub PropertyCallsExample(ByVal elementList As AutomationElement)
    ' The following two calls are equivalent.
    Dim name As String = elementList.Current.Name
    name = CStr(elementList.GetCurrentPropertyValue(AutomationElement.NameProperty))

    ' The following shows how to ignore the default property, which
    '  would probably be an empty string if the property is not supported.
    '  Passing "false" as the second parameter is equivalent to using the overload
    '  that does not have this parameter.
    Dim help As Object = elementList.GetCurrentPropertyValue(AutomationElement.HelpTextProperty, True)
    If help Is AutomationElement.NotSupported Then
        help = "No help available"
    End If
    Dim helpText As String = CStr(help)

End Sub
```

See also

- UI Automation Properties for Clients
- Use Caching in UI Automation
- Caching in UI Automation Clients

Use Caching in UI Automation

12/4/2019 • 7 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section shows how to implement caching of AutomationElement properties and control patterns.

Activate a Cache Request

1. Create a CacheRequest.
2. Specify properties and patterns to cache by using Add.
3. Specify the scope of caching by setting the TreeScope property.
4. Specify the view of the subtree by setting the TreeFilter property.
5. Set the AutomationElementMode property to None if you wish to increase efficiency by not retrieving a full reference to objects. (This will make it impossible to retrieve current values from those objects.)
6. Activate the request by using Activate within a `using` block (`Using` in Microsoft Visual Basic .NET).

After obtaining AutomationElement objects or subscribing to events, deactivate the request by using Pop (if Push was used) or by disposing the object created by Activate. (Use Activate in a `using` block (`Using` in Microsoft Visual Basic .NET).

Cache AutomationElement Properties

1. While a CacheRequest is active, obtain AutomationElement objects by using FindFirst or FindAll; or obtain an AutomationElement as the source of an event that you registered for when the CacheRequest was active. (You can also create a cache by passing a CacheRequest to GetUpdatedCache or one of the TreeWalker methods.)
2. Use GetCachedPropertyValue or retrieve a property from the Cached property of the AutomationElement.

Obtain Cached Patterns and Their Properties

1. While a CacheRequest is active, obtain AutomationElement objects by using FindFirst or FindAll; or obtain an AutomationElement as the source of an event that you registered for when the CacheRequest was active. (You can also create a cache by passing a CacheRequest to GetUpdatedCache or one of the TreeWalker methods.)
2. Use GetCachedPattern or TryGetCachedPattern to retrieve a cached pattern.
3. Retrieve property values from the `Cached` property of the control pattern.

Example

The following code example shows various aspects of caching, using Activate to activate the CacheRequest.

```
/// <summary>
/// Caches and retrieves properties for a list item by using CacheRequest.Activate.
/// </summary>
/// <param name="elementList">Element from which to retrieve a child element.</param>
/// <remarks>
/// This code demonstrates various aspects of caching. It is not intended to be
/// an example of a useful method.
/// </remarks>
private void CachePropertiesByActivate(AutomationElement elementList)
{
    AutomationElement elementListItem;

    // Set up the request.
    CacheRequest cacheRequest = new CacheRequest();
    cacheRequest.Add(AutomationElement.NameProperty);
    cacheRequest.Add(AutomationElement.IsEnabledProperty);
    cacheRequest.Add(SelectionItemPattern.Pattern);
    cacheRequest.Add(SelectionItemPattern.SelectionContainerProperty);

    // Obtain an element and cache the requested items.
    using (cacheRequest.Activate())
    {
        Condition cond = new PropertyCondition(AutomationElement.IsSelectionItemPatternAvailableProperty,
true);
        elementListItem = elementList.FindFirst(TreeScope.Children, cond);
    }
    // The CacheRequest is now inactive.

    // Retrieve the cached property and pattern.
    SelectionItemPattern pattern;
    String itemName;
    try
    {
        itemName = elementListItem.Cached.Name;
        pattern = elementListItem.GetCachedPattern(SelectionItemPattern.Pattern) as SelectionItemPattern;
    }
    catch (InvalidOperationException)
    {
        Console.WriteLine("Object was not in cache.");
        return;
    }
    // Alternatively, you can use TryGetCachedPattern to retrieve the cached pattern.
    object cachedPattern;
    if (true == elementListItem.TryGetCachedPattern(SelectionItemPattern.Pattern, out cachedPattern))
    {
        pattern = cachedPattern as SelectionItemPattern;
    }

    // Specified pattern properties are also in the cache.
    AutomationElement parentList = pattern.Cached.SelectionContainer;

    // The following line will raise an exception, because the HelpText property was not cached.
    /*** String itemHelp = elementListItem.Cached.HelpText; ***/

    // Similarly, pattern properties that were not specified in the CacheRequest cannot be
    // retrieved from the cache. This would raise an exception.
    /*** bool selected = pattern.Cached.IsSelected; ***/

    // This is still a valid call, even though the property is in the cache.
    // Of course, the cached value and the current value are not guaranteed to be the same.
    itemName = elementListItem.Current.Name;
}
```

```
''' <summary>
''' Caches and retrieves properties for a list item by using CacheRequest.Activate.
''' </summary>
''' <param name="elementList">Element from which to retrieve a child element.</param>
''' <remarks>
''' This code demonstrates various aspects of caching. It is not intended to be
''' an example of a useful method.
''' </remarks>
Private Sub CachePropertiesByActivate(ByVal elementList As AutomationElement)

    ' Set up the request.
    Dim myCacheRequest As New CacheRequest()
    myCacheRequest.Add(AutomationElement.NameProperty)
    myCacheRequest.Add(AutomationElement.IsEnabledProperty)
    myCacheRequest.Add(SelectionItemPattern.Pattern)
    myCacheRequest.Add(SelectionItemPattern.SelectionContainerProperty)

    Dim elementListItem As AutomationElement

    ' Obtain an element and cache the requested items.
    Using myCacheRequest.Activate()
        Dim myCondition As New PropertyCondition( _
            AutomationElement.IsSelectionItemPatternAvailableProperty, True)
        elementListItem = elementList.FindFirst(TreeScope.Children, myCondition)
    End Using


    ' The CacheRequest is now inactive.
    ' Retrieve the cached property and pattern.
    Dim pattern As SelectionItemPattern
    Dim itemName As String
    Try
        itemName = elementListItem.Cached.Name
        pattern = DirectCast(elementListItem.GetCachedPattern(SelectionItemPattern.Pattern), _
            SelectionItemPattern)
    Catch ex As InvalidOperationException
        Console.WriteLine("Object was not in cache.")
        Return
    End Try
    ' Alternatively, you can use TryGetCachedPattern to retrieve the cached pattern.
    Dim cachedPattern As Object = Nothing
    If True = elementListItem.TryGetCachedPattern(SelectionItemPattern.Pattern, cachedPattern) Then
        pattern = DirectCast(cachedPattern, SelectionItemPattern)
    End If

    ' Specified pattern properties are also in the cache.
    Dim parentList As AutomationElement = pattern.Cached.SelectionContainer

    ' The following line will raise an exception, because the HelpText property was not cached.
    '** String itemHelp = elementListItem.Cached.HelpText; **

    ' Similarly, pattern properties that were not specified in the CacheRequest cannot be
    ' retrieved from the cache. This would raise an exception.
    '** bool selected = pattern.Cached.IsSelected; **

    ' This is still a valid call, even though the property is in the cache.
    ' Of course, the cached value and the current value are not guaranteed to be the same.
    itemName = elementListItem.Current.Name
End Sub
```

Example

The following code example shows various aspects of caching, using Push to activate the CacheRequest. Except when you wish to nest cache requests, it is preferable to use Activate.

```
/// <summary>
/// Caches and retrieves properties for a list item by using CacheRequest.Push.
/// </summary>
/// <param name="autoElement">Element from which to retrieve a child element.</param>
/// <remarks>
/// This code demonstrates various aspects of caching. It is not intended to be
/// an example of a useful method.
/// </remarks>
private void CachePropertiesByPush(AutomationElement elementList)
{
    // Set up the request.
    CacheRequest cacheRequest = new CacheRequest();

    // Do not get a full reference to the cached objects, only to their cached properties and patterns.
    cacheRequest.AutomationElementMode = AutomationElementMode.None;

    // Cache all elements, regardless of whether they are control or content elements.
    cacheRequest.TreeFilter = Automation.RawViewCondition;

    // Property and pattern to cache.
    cacheRequest.Add(AutomationElement.NameProperty);
    cacheRequest.Add(SelectionItemPattern.Pattern);

    // Activate the request.
    cacheRequest.Push();

    // Obtain an element and cache the requested items.
    Condition cond = new PropertyCondition(AutomationElement.IsSelectionItemPatternAvailableProperty, true);
    AutomationElement elementListItem = elementList.FindFirst(TreeScope.Children, cond);

    // At this point, you could call another method that creates a CacheRequest and calls Push/Pop.
    // While that method was retrieving automation elements, the CacheRequest set in this method
    // would not be active.

    // Deactivate the request.
    cacheRequest.Pop();

    // Retrieve the cached property and pattern.
    String itemName = elementListItem.Cached.Name;
    SelectionItemPattern pattern = elementListItem.GetCachedPattern(SelectionItemPattern.Pattern) as
SelectionItemPattern;

    // The following is an alternative way of retrieving the Name property.
    itemName = elementListItem.GetCachedPropertyValue(AutomationElement.NameProperty) as String;

    // This is yet another way, which returns AutomationElement.NotSupported if the element does
    // not supply a value. If the second parameter is false, a default name is returned.
    object objName = elementListItem.GetCachedPropertyValue(AutomationElement.NameProperty, true);
    if (objName == AutomationElement.NotSupported)
    {
        itemName = "Unknown";
    }
    else
    {
        itemName = objName as String;
    }

    // The following call raises an exception, because only the cached properties are available,
    //  as specified by cacheRequest.AutomationElementMode. If AutomationElementMode had its
    //  default value (Full), this call would be valid.
    /*** bool enabled = elementListItem.Current.IsEnabled; ***/
}
```

```
''' <summary>
''' Caches and retrieves properties for a list item by using CacheRequest.Push.
''' </summary>
''' <param name="elementList">Element from which to retrieve a child element.</param>
''' <remarks>
''' This code demonstrates various aspects of caching. It is not intended to be
''' an example of a useful method.
''' </remarks>
Private Sub CachePropertiesByPush(ByVal elementList As AutomationElement)
    ' Set up the request.
    Dim cacheRequest As New CacheRequest()

    ' Do not get a full reference to the cached objects, only to their cached properties and patterns.
    cacheRequest.AutomationElementMode = AutomationElementMode.None

    ' Cache all elements, regardless of whether they are control or content elements.
    cacheRequest.TreeFilter = Automation.RawViewCondition

    ' Property and pattern to cache.
    cacheRequest.Add(AutomationElement.NameProperty)
    cacheRequest.Add(SelectionItemPattern.Pattern)

    ' Activate the request.
    cacheRequest.Push()

    ' Obtain an element and cache the requested items.
    Dim myCondition As New PropertyCondition(AutomationElement.IsSelectionItemPatternAvailableProperty, _
        True)
    Dim elementListItem As AutomationElement = elementList.FindFirst(TreeScope.Children, myCondition)

    ' At this point, you could call another method that creates a CacheRequest and calls Push/Pop.
    ' While that method was retrieving automation elements, the CacheRequest set in this method
    ' would not be active.
    ' Deactivate the request.
    cacheRequest.Pop()

    ' Retrieve the cached property and pattern.
    Dim itemName As String = elementListItem.Cached.Name
    Dim pattern As SelectionItemPattern = _
        DirectCast(elementListItem.GetCachedPattern(SelectionItemPattern.Pattern), SelectionItemPattern)

    ' The following is an alternative way of retrieving the Name property.
    itemName = CStr(elementListItem.GetCachedPropertyValue(AutomationElement.NameProperty))

    ' This is yet another way, which returns AutomationElement.NotSupported if the element does
    ' not supply a value. If the second parameter is false, a default name is returned.
    Dim objName As Object = elementListItem.GetCachedPropertyValue(AutomationElement.NameProperty, True)
    If objName Is AutomationElement.NotSupported Then
        itemName = "Unknown"
    Else
        itemName = CStr(objName)
    End If
    ' The following call raises an exception, because only the cached properties are available,
    '  as specified by cacheRequest.AutomationElementMode. If AutomationElementMode had its
    '  default value (Full), this call would be valid.
    '** bool enabled = elementListItem.Current.IsEnabled; **

End Sub
```

See also

- Caching in UI Automation Clients

Subscribe to UI Automation Events

11/23/2019 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to subscribe to events raised by UI Automation providers.

Example

The following example code registers an event handler for the event that is raised when a control such as a button is invoked, and removes it when the application form closes. The event is identified by an AutomationEvent passed as a parameter to AddAutomationEventHandler.

```
// Member variables.
AutomationElement ElementSubscribeButton;
AutomationEventHandler UIAeventHandler;

/// <summary>
/// Register an event handler for InvokedEvent on the specified element.
/// </summary>
/// <param name="elementButton">The automation element.</param>
public void SubscribeToInvoke(AutomationElement elementButton)
{
    if (elementButton != null)
    {
        Automation.AddAutomationEventHandler(InvokePattern.InvokedEvent,
             elementButton, TreeScope.Element,
             UIAeventHandler = new AutomationEventHandler(OnUIAutomationEvent));
        ElementSubscribeButton = elementButton;
    }
}

/// <summary>
/// AutomationEventHandler delegate.
/// </summary>
/// <param name="src">Object that raised the event.</param>
/// <param name="e">Event arguments.</param>
private void OnUIAutomationEvent(object src, AutomationEventArgs e)
{
    // Make sure the element still exists. Elements such as tooltips
    // can disappear before the event is processed.
    AutomationElement sourceElement;
    try
    {
        sourceElement = src as AutomationElement;
    }
    catch (ElementNotAvailableException)
    {
        return;
    }
    if (e.EventId == InvokePattern.InvokedEvent)
    {
        // TODO Add handling code.
    }
    else
    {
        // TODO Handle any other events that have been subscribed to.
    }
}

private void ShutdownUIA()
{
    if (UIAeventHandler != null)
    {
        Automation.RemoveAutomationEventHandler(InvokePattern.InvokedEvent,
            ElementSubscribeButton, UIAeventHandler);
    }
}
```

```
' Member variables.
Private ElementSubscribeButton As AutomationElement
Private UIAeventHandler As AutomationEventHandler


''' <summary>
''' Register an event handler for InvokedEvent on the specified element.
''' </summary>
''' <param name="elementButton">The automation element.</param>
Public Sub SubscribeToInvoke(ByVal elementButton As AutomationElement)
    If (elementButton IsNot Nothing) Then
        UIAeventHandler = New AutomationEventHandler(AddressOf OnUIAutomationEvent)
        Automation.AddAutomationEventHandler(InvokePattern.InvokedEvent, elementButton, _
        TreeScope.Element, UIAeventHandler)
        ElementSubscribeButton = elementButton
    End If

End Sub


''' <summary>
''' AutomationEventHandler delegate.
''' </summary>
''' <param name="src">Object that raised the event.</param>
''' <param name="e">Event arguments.</param>
Private Sub OnUIAutomationEvent(ByVal src As Object, ByVal e As AutomationEventArgs)
    ' Make sure the element still exists. Elements such as tooltips can disappear
    ' before the event is processed.
    Dim sourceElement As AutomationElement
    Try
        sourceElement = DirectCast(src, AutomationElement)
    Catch ex As ElementNotAvailableException
        Exit Sub
    End Try
    If e.EventId Is InvokePattern.InvokedEvent Then
        ' TODO Add handling code.
    Else
    End If
    ' TODO Handle any other events that have been subscribed to.
    Console.WriteLine("Event: " & e.EventId.ProgrammaticName)
End Sub

Private Sub ShutdownUIA()
    If (UIAeventHandler IsNot Nothing) Then
        Automation.RemoveAutomationEventHandler(InvokePattern.InvokedEvent, ElementSubscribeButton,
UIAeventHandler)
    End If

End Sub
```

Example

The following example shows how to use Microsoft UI Automation to subscribe to an event that is raised when the focus changes. The event handler is unregistered in a method that could be called on application shutdown, or when notification of UI events is no longer required.

```
AutomationFocusChangedEventHandler focusHandler = null;

/// <summary>
/// Create an event handler and register it.
/// </summary>
public void SubscribeToFocusChange()
{
    focusHandler = new AutomationFocusChangedEventHandler(OnFocusChange);
    Automation.AddAutomationFocusChangedEventHandler(focusHandler);
}

/// <summary>
/// Handle the event.
/// </summary>
/// <param name="src">Object that raised the event.</param>
/// <param name="e">Event arguments.</param>
private void OnFocusChange(object src, AutomationFocusChangedEventArgs e)
{
    // TODO Add event handling code.
    // The arguments tell you which elements have lost and received focus.
}

/// <summary>
/// Cancel subscription to the event.
/// </summary>
public void UnsubscribeFocusChange()
{
    if (focusHandler != null)
    {
        Automation.RemoveAutomationFocusChangedEventHandler(focusHandler);
    }
}
```

```
Private focusHandler As AutomationFocusChangedEventHandler = Nothing


''' <summary>
''' Create an event handler and register it.
''' </summary>
Public Sub SubscribeToFocusChange()
    focusHandler = New AutomationFocusChangedEventHandler(AddressOf OnFocusChange)
    Automation.AddAutomationFocusChangedEventHandler(focusHandler)

End Sub


''' <summary>
''' Handle the event.
''' </summary>
''' <param name="src">Object that raised the event.</param>
''' <param name="e">Event arguments.</param>
Private Sub OnFocusChange(ByVal src As Object, ByVal e As AutomationFocusChangedEventArgs)

End Sub

' TODO Add event handling code.
' The arguments tell you which elements have lost and received focus.

''' <summary>
''' Cancel subscription to the event.
''' </summary>
Public Sub UnsubscribeFocusChange()
    If (focusHandler IsNot Nothing) Then
        Automation.RemoveAutomationFocusChangedEventHandler(focusHandler)
    End If

End Sub
```

See also

- AddAutomationEventHandler
- RemoveAllEventHandlers
- RemoveAutomationEventHandler
- UI Automation Events Overview

Register a Client-Side Provider Assembly

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to register a DLL that contains client-side UI Automation providers.

Example

The following example shows how to register an assembly that contains a provider for a console window.

```
using System;
using System.Windows.Automation;
using System.Reflection;
using System.Runtime.InteropServices;
using System.IO;

namespace CSClient
{
    class CSClientProgram
    {
        [DllImport("kernel32.dll")]
        static extern IntPtr GetConsoleWindow();

        static void Main(string[] args)
        {
            // TODO  Change the path to the appropriate one for your CSProviderDLL.
            string fileloc = @"C:\SampleDependencies\CSProviderDLL.dll";
            Assembly a = null;
            try
            {
                a = Assembly.LoadFile(fileloc);
            }
            catch (FileNotFoundException e1)
            {
                Console.WriteLine(e1.Message);
            }
            if (a != null)
            {
                try
                {
                    ClientSettings.RegisterClientSideProviderAssembly(a.GetName());
                }
                catch (ProxyAssemblyNotLoadedException e)
                {
                    Console.WriteLine(e.Message);
                }

                IntPtr hwnd = GetConsoleWindow();

                // Get an AutomationElement that represents the window.
                AutomationElement elementWindow = AutomationElement.FromHandle(hwnd);
                Console.WriteLine("Found UI Automation client-side provider for:");

                // The name property is furnished by the client-side provider.
                Console.WriteLine(elementWindow.Current.Name);
                Console.WriteLine();
            }
            Console.WriteLine("Press any key to exit.");
            Console.ReadLine();
        }
    }
}
```

```
Imports System.Windows.Automation
Imports System.Reflection
Imports System.Runtime.InteropServices
Imports System.IO


Namespace CSClient
    Friend Class CSClientProgram
        <DllImport("kernel32.dll")>
        Shared Function GetConsoleWindow() As IntPtr
        End Function

        Shared Sub Main(ByVal args() As String)
            ' TODO  Change the path to the appropriate one for your CSProviderDLL.
            Dim fileloc As String = "C:\SampleDependencies\CSProviderDLL.dll"
            Dim a As System.Reflection.Assembly = Nothing
            Try
                a = System.Reflection.Assembly.LoadFile(fileloc)
            Catch e1 As FileNotFoundException
                Console.WriteLine(e1.Message)

            End Try
            If a IsNot Nothing Then
                Try
                    ClientSettings.RegisterClientSideProviderAssembly(a.GetName())
                Catch e As ProxyAssemblyNotLoadedException
                    Console.WriteLine(e.Message)
                End Try

                Dim hwnd As IntPtr = GetConsoleWindow()

                ' Get an AutomationElement that represents the window.
                Dim elementWindow As AutomationElement = AutomationElement.FromHandle(hwnd)
                Console.WriteLine("Found UI Automation client-side provider for:")

                ' The name property is furnished by the client-side provider.
                Console.WriteLine(elementWindow.Current.Name)
                Console.WriteLine()
            End If
            Console.WriteLine("Press any key to exit.")
            Console.ReadLine()
        End Sub
    End Class
End Namespace
```

See also

- Create a Client-Side UI Automation Provider

Use the AutomationID Property

1/8/2020 • 9 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains scenarios and sample code that show how and when the AutomationIdProperty can be used to locate an element within the UI Automation tree.

AutomationIdProperty uniquely identifies a UI Automation element from its siblings. For more information on property identifiers related to control identification, see UI Automation Properties Overview.

NOTE

AutomationIdProperty does not guarantee a unique identity throughout the tree; it typically needs container and scope information to be useful. For example, an application may contain a menu control with multiple top-level menu items that, in turn, have multiple child menu items. These secondary menu items may be identified by a generic scheme such as "Item1", "Item 2", and so on, allowing duplicate identifiers for children across top-level menu items.

Scenarios

Three primary UI Automation client application scenarios have been identified that require the use of AutomationIdProperty to achieve accurate and consistent results when searching for elements.

NOTE

AutomationIdProperty is supported by all UI Automation elements in the control view except top-level application windows, UI Automation elements derived from Windows Presentation Foundation (WPF) controls that do not have an ID or x:Uid, and UI Automation elements derived from Win32 controls that do not have a control ID.

Use a unique and discoverable AutomationID to locate a specific element in the UI Automation tree

- Use a tool such as UI Spy to report the AutomationIdProperty of a UI element of interest. This value can then be copied and pasted into a client application such as a test script for subsequent automated testing. This approach reduces and simplifies the code necessary to identify and locate an element at runtime.

Caution

In general, you should try to obtain only direct children of the RootElement. A search for descendants may iterate through hundreds or even thousands of elements, possibly resulting in a stack overflow. If you are attempting to obtain a specific element at a lower level, you should start your search from the application window or from a container at a lower level.

```
///--------------------------------------------------------------------
/// <summary>
/// Finds all elements in the UI Automation tree that have a specified
/// AutomationID.
/// </summary>
/// <param name="targetApp">
/// The root element from which to start searching.
/// </param>
/// <param name="automationID">
/// The AutomationID value of interest.
/// </param>
/// <returns>
/// The collection of UI Automation elements that have the specified
/// AutomationID value.
/// </returns>
///--------------------------------------------------------------------
private AutomationElementCollection FindElementFromAutomationID(AutomationElement targetApp,
    string automationID)
{
    return targetApp.FindAll(
        TreeScope.Descendants,
        new PropertyCondition(AutomationElement.AutomationIdProperty, automationID));
}
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Finds all elements in the UI Automation tree that have a specified
''' AutomationID.
''' </summary>
''' <param name="targetApp">
''' The root element from which to start searching.
''' </param>
''' <param name="automationID">
''' The AutomationID value of interest.
''' </param>
''' <returns>
''' The collection of automation elements that have the specified
''' AutomationID value.
''' </returns>
'''--------------------------------------------------------------------
Private Function FindElementFromAutomationID( _
ByVal targetApp As AutomationElement, _
ByVal automationID As String) As AutomationElementCollection
    Return targetApp.FindAll( _
    TreeScope.Descendants, _
    New PropertyCondition( _
    AutomationElement.AutomationIdProperty, automationID))
End Function 'FindElementFromAutomationID
```

Use a persistent path to return to a previously identified AutomationElement

- Client applications, from simple test scripts to robust record and playback utilities, may require access to elements that are not currently instantiated, such as a file open dialog or a menu item and therefore do not exist in the UI Automation tree. These elements can only be instantiated by reproducing, or "playing back", a specific sequence of UI actions through the use of UI Automation properties such as AutomationID, control patterns, and event listeners.

```
///--------------------------------------------------------------------
/// <summary>
/// Creates a UI Automation thread.
/// </summary>
/// <param name="sender">Object that raised the event.</param>
/// <param name="e">Event arguments.</param>
/// <remarks>
/// UI Automation must be called on a separate thread if the client
/// application itself could become a target for event handling.
/// For example, focus tracking is a desktop event that could involve
/// the client application.
/// </remarks>
///--------------------------------------------------------------------
private void CreateUIAThread(object sender, EventArgs e)
{
    // Start another thread to do the UI Automation work.
    ThreadStart threadDelegate = new ThreadStart(CreateUIAWorker);
    Thread workerThread = new Thread(threadDelegate);
    workerThread.Start();
}

///--------------------------------------------------------------------
/// <summary>
/// Delegated method for ThreadStart. Creates a UI Automation worker
/// class that does all UI Automation related work.
/// </summary>
///--------------------------------------------------------------------
public void CreateUIAWorker()
{
   uiautoWorker = new FindByAutomationID(targetApp);
}
private FindByAutomationID uiautoWorker;
```

```vb
'''--------------------------------------------------------------------
''' <summary>
''' Creates a UI Automation thread.
''' </summary>
''' <param name="sender">Object that raised the event.</param>
''' <param name="e">Event arguments.</param>
''' <remarks>
''' UI Automation must be called on a separate thread if the client
''' application itself could become a target for event handling.
''' For example, focus tracking is a desktop event that could involve
''' the client application.
''' </remarks>
'''--------------------------------------------------------------------
Private Sub CreateUIAThread(ByVal sender As Object, ByVal e As EventArgs)

    ' Start another thread to do the UI Automation work.
    Dim threadDelegate As New ThreadStart(AddressOf CreateUIAWorker)
    Dim workerThread As New Thread(threadDelegate)
    workerThread.Start()

End Sub


'''--------------------------------------------------------------------
''' <summary>
''' Delegated method for ThreadStart. Creates a UI Automation worker
''' class that does all UI Automation related work.
''' </summary>
'''--------------------------------------------------------------------
Public Sub CreateUIAWorker()

    uiautoWorker = New UIAWorker(targetApp)

End Sub

Private uiautoWorker As UIAWorker
```

```csharp
///--------------------------------------------------------------------
/// <summary>
/// Function to playback through a series of recorded events calling
/// a WriteToScript function for each event of interest.
/// </summary>
/// <remarks>
/// A major drawback to using AutomationID for recording user
/// interactions in a volatile UI is the probability of catastrophic
/// change in the UI. For example, the //Processes// dialog where items
/// in the listbox container can change with no input from the user.
/// This mandates thtat a record and playback application must be
/// reliant on the tester owning the UI being tested. In other words,
/// there has to be a contract between the provider and client that
/// excludes uncontrolled, external applications. The added benefit
/// is the guarantee that each control in the UI should have an
/// AutomationID assigned to it.
///
/// This function relies on a UI Automation worker class to create
/// the System.Collections.Generic.Queue object that stores the
/// information for the recorded user interactions. This
/// allows post-processing of the recorded items prior to actually
/// writing them to a script. If this is not necessary the interaction
/// could be written to the script immediately.
/// </remarks>
///--------------------------------------------------------------------
private void Playback(AutomationElement targetApp)
{
    AutomationElement element;
    foreach(ElementStore storedItem in uiautoWorker.elementQueue)
    {
        PropertyCondition propertyCondition =
            new PropertyCondition(
            AutomationElement.AutomationIdProperty, storedItem.AutomationID);
        // Confirm the existence of a control.
        // Depending on the controls and complexity of interaction
        // this step may not be necessary or may require additional
        // functionality. For example, to confirm the existence of a
        // child menu item that had been invoked the parent menu item
        // would have to be expanded.
        element = targetApp.FindFirst(TreeScope.Descendants, propertyCondition);
        if(element == null)
        {
            // Control not available, unable to continue.
            // TODO: Handle error condition.
            return;
        }
        WriteToScript(storedItem.AutomationID, storedItem.EventID);
    }
}

///--------------------------------------------------------------------
/// <summary>
/// Generates script code and outputs the code to a text control in
/// the client.
/// </summary>
/// <param name="automationID">
/// The AutomationID of the current control.
/// </param>
/// <param name="eventID">
/// The event recorded on that control.
/// </param>
///--------------------------------------------------------------------
private void WriteToScript(string automationID, string eventID)
{
    // Script code would be generated and written to an output file
    // as plain text at this point, but for the
    // purposes of this example we just write to the console.
    Console.WriteLine(automationID + " - " + eventID);
}
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Function to playback through a series of recorded events calling
''' a WriteToScript function for each event of interest.
''' </summary>
''' <remarks>
''' A major drawback to using AutomationID for recording user
''' interactions in a volatile UI is the probability of catastrophic
''' change in the UI. For example, the 'Processes' dialog where items
''' in the listbox container can change with no input from the user.
''' This mandates thtat a record and playback application must be
''' reliant on the tester owning the UI being tested. In other words,
''' there has to be a contract between the provider and client that
''' excludes uncontrolled, external applications. The added benefit
''' is the guarantee that each control in the UI should have an
''' AutomationID assigned to it.
'''
''' This function relies on a UI Automation worker class to create
''' the System.Collections.Generic.Queue object that stores the
''' information for the recorded user interactions. This
''' allows post-processing of the recorded items prior to actually
''' writing them to a script. If this is not necessary the interaction
''' could be written to the script immediately.
''' </remarks>
'''--------------------------------------------------------------------
```

```
Private Sub Playback(ByVal targetApp As AutomationElement)

    Dim element As AutomationElement
    Dim storedItem As ElementStore
    For Each storedItem In uiautoWorker.elementQueue
        Dim propertyCondition As New PropertyCondition( _
        AutomationElement.AutomationIdProperty, storedItem.AutomationID)
        ' Confirm the existence of a control.
        ' Depending on the controls and complexity of interaction
        ' this step may not be necessary or may require additional
        ' functionality. For example, to confirm the existence of a
        ' child menu item that had been invoked the parent menu item
        ' would have to be expanded.
        element = targetApp.FindFirst( _
        TreeScope.Descendants, propertyCondition)
        If element Is Nothing Then
            ' Control not available, unable to continue.
            ' TODO: Handle error condition.
            Return
        End If
        WriteToScript(storedItem.AutomationID, storedItem.EventID)
    Next storedItem

End Sub


'''--------------------------------------------------------------------
''' <summary>
''' Generates script code and outputs the code to a text control in
''' the client.
''' </summary>
''' <param name="automationID">
''' The AutomationID of the current control.
''' </param>
''' <param name="eventID">
''' The event recorded on that control.
''' </param>
'''--------------------------------------------------------------------
Private Sub WriteToScript( _
ByVal automationID As String, ByVal eventID As String)

    ' Script code would be generated and written to an output file
    ' as plain text at this point, but for the
    ' purposes of this example we just write to the console.
    Console.WriteLine(automationID + " - " + eventID)

End Sub
```

Use a relative path to return to a previously identified AutomationElement

- In certain circumstances, since AutomationID is only guaranteed to be unique amongst siblings, multiple elements in the UI Automation tree may have identical AutomationID property values. In these situations the elements can be uniquely identified based on a parent and, if necessary, a grandparent. For example, a developer may provide a menu bar with multiple menu items each with multiple child menu items where the children are identified with sequential AutomationID's such as "Item1", "Item2", and so on. Each menu item could then be uniquely identified by its AutomationID along with the AutomationID of its parent and, if necessary, its grandparent.

See also

- AutomationIdProperty
- UI Automation Tree Overview
- Find a UI Automation Element Based on a Property Condition

UI Automation Control Patterns

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section provides detailed information about how to use Microsoft UI Automation control patterns.

In This Section

Implementing the UI Automation Dock Control Pattern

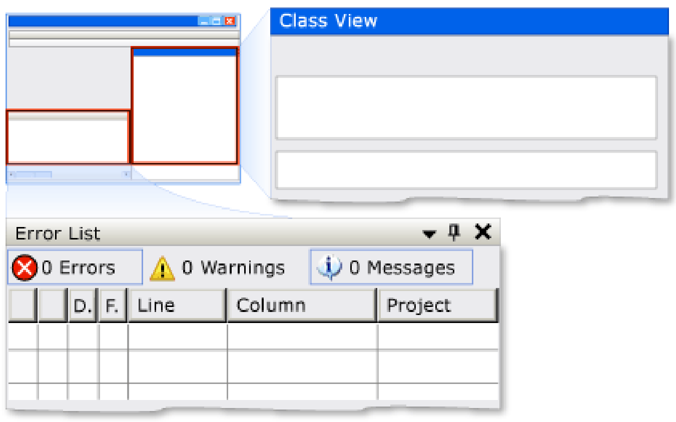
3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IDockProvider, including information about properties. Links to additional references are listed at the end of the topic.

The DockPattern control pattern is used to expose the dock properties of a control within a docking container. A docking container is a control that allows you to arrange child elements horizontally and vertically, relative to each other. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.



Docking Example from Visual Studio Where "Class View" Window Is DockPosition.Right and "Error List" Window Is DockPosition.Bottom

Implementation Guidelines and Conventions

When implementing the Dock control pattern, note the following guidelines and conventions:

- IDockProvider does not expose any properties of the docking container or any properties of controls that are docked adjacent to the current control within the docking container.
- Controls are docked relative to each other based on their current z-order; the higher their z-order placement, the farther they are placed from the specified edge of the docking container.
- If the docking container is resized, any docked controls within the container will be repositioned flush to the same edge to which they were originally docked. The docked controls will also resize to fill any space within the container according to the docking behavior of their DockPosition. For example, if Top is specified, the left and right sides of the control will expand to fill any available space. If Fill is specified, all four sides of the control will expand to fill any available space.
- On a multi-monitor system, controls should dock to the left or right side of the current monitor. If that is not possible, they should dock to the left side of the leftmost monitor or the right side of the rightmost monitor.

Required Members for IDockProvider

The following properties and methods are required for implementing the IDockProvider interface.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
DockPosition	Property	None
SetDockPosition	Method	None

This control pattern has no associated events.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
InvalidOperationException	SetDockPosition - When a control is not able to execute the requested dock style.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation ExpandCollapse Control Pattern

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IExpandCollapseProvider, including information about properties, methods, and events. Links to additional references are listed at the end of the overview.

The ExpandCollapsePattern control pattern is used to support controls that visually expand to display more content and collapse to hide content. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the ExpandCollapse control pattern, note the following guidelines and conventions:

- Aggregate controls—built with child objects that provide the UI with expand/collapse functionality—must support the ExpandCollapsePattern control pattern whereas their child elements do not. For example, a combo box control is built with a combination of list box, button, and edit controls, but it is only the parent combo box that must support the ExpandCollapsePattern.

  NOTE

  An exception is the menu control, which is an aggregate of individual MenuItem objects. The MenuItem objects can support the ExpandCollapsePattern control pattern, but the parent Menu control cannot. A similar exception applies to the Tree and Tree Item controls.

- When the ExpandCollapseState of a control is set to LeafNode, any ExpandCollapsePattern functionality is currently inactive for the control and the only information that can be obtained using this control pattern is the ExpandCollapseState. If any child objects are subsequently added, the ExpandCollapseState changes and ExpandCollapsePattern functionality is activated.
- ExpandCollapseState refers to the visibility of immediate child objects only; it does not refer to the visibility of all descendant objects.
- Expand and Collapse functionality is control-specific. The following are examples of this behavior.
  - The Office Personal Menu can be a tri-state MenuItem (Expanded, Collapsed and PartiallyExpanded) where the control specifies the state to adopt when an Expand or Collapse is called.
  - Calling Expand on a TreeItem may display all descendants or only immediate children.
  - If calling Expand or Collapse on a control maintains the state of its descendants, a visibility change event should be sent, not a state change event If the parent control does not maintain the state of its descendants when collapsed, the control may destroy all the descendants that are no longer visible

and raise a destroyed event; or it may change the ExpandCollapseState for each descendant and raise a visibility change event.

- To guarantee navigation, it is desirable for an object to be in the UI Automation tree (with appropriate visibility state) regardless of its parents ExpandCollapseState. If descendants are generated on demand, they may only appear in the UI Automation tree after being displayed for the first time or only while they are visible.

Required Members for IExpandCollapseProvider

The following properties and methods are required for implementing IExpandCollapseProvider.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
ExpandCollapseState	Property	None
Expand	Method	None
Collapse	Method	None
AutomationPropertyChangedEventHandler	Event	This control has no associated events; use this generic delegate.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
InvalidOperationException	Either Expand or Collapse is called when the ExpandCollapseState = LeafNode.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Navigate Among UI Automation Elements with TreeWalker
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Grid Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

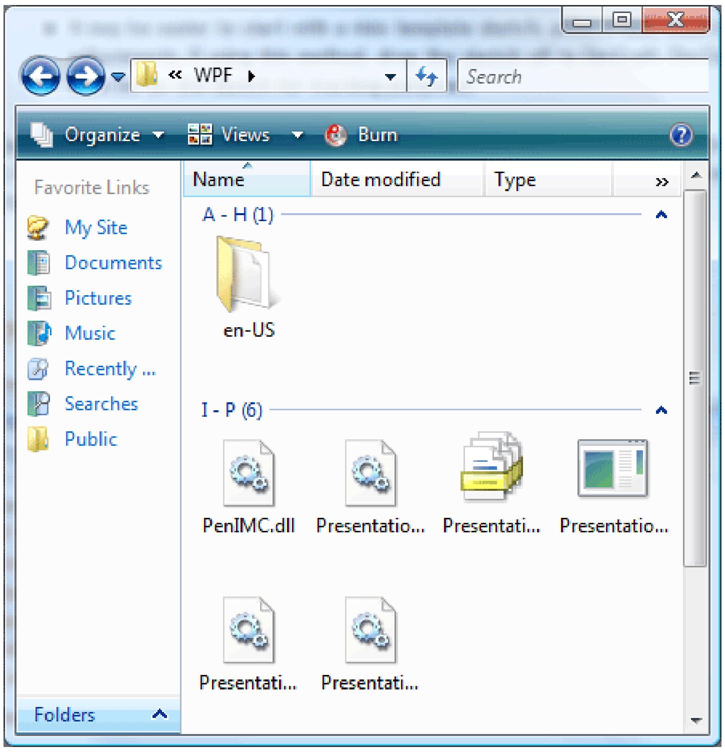
This topic introduces guidelines and conventions for implementing IGridProvider, including information about properties, methods, and events. Links to additional references are listed at the end of the overview.

The GridPattern control pattern is used to support controls that act as containers for a collection of child elements. The children of this element must implement IGridItemProvider and be organized in a two-dimensional logical coordinate system that can be traversed by row and column. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Grid control pattern, note the following guidelines and conventions:

- Grid coordinates are zero-based with the upper left (or upper right cell depending on locale) having coordinates (0, 0).
- If a cell is empty, a UI Automation element must still be returned in order to support the ContainingGrid property for that cell. This is possible when the layout of child elements in the grid is similar to a ragged array (see example below).



Example of a Grid Control with Empty Coordinates

- A grid with a single item is still required to implement IGridProvider if it is logically considered to be a grid. The number of child items in the grid is immaterial.
- Hidden rows and columns, depending on the provider implementation, may be loaded in the UI Automation tree and therefore will be reflected in the RowCount and ColumnCount properties. If the hidden rows and columns have not yet been loaded, they should not be counted.
- IGridProvider does not enable active manipulation of a grid; ITransformProvider must be implemented to enable this functionality.
- Use a StructureChangedEventHandler to listen for structural or layout changes to the grid such as cells that have been added, removed, or merged.
- Use a AutomationFocusChangedEventHandler to track traversal through the items or cells of a grid.

Required Members for IGridProvider

The following properties and methods are required for implementing the IGridProvider interface.

REQUIRED MEMBERS	TYPE	NOTES
RowCount	Property	None
ColumnCount	Property	None
GetItem	Method	None

This control pattern has no associated events.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
ArgumentOutOfRangeException	GetItem - If the requested row coordinate is larger than the RowCount or the column coordinate is larger than the ColumnCount.
ArgumentOutOfRangeException	GetItem - If either of the requested row or column coordinates is less than zero.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Implementing the UI Automation GridItem Control Pattern
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation GridItem Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IGridItemProvider, including information about properties. Links to additional references are listed at the end of the overview.

The GridItemPattern control pattern is used to support individual child controls of containers that implement IGridProvider. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing IGridProvider, note the following guidelines and conventions:

- Grid coordinates are zero-based with the upper left cell having coordinates (0, 0).
- Merged cells will report their Row and Column properties based on their underlying anchor cell as defined by the UI Automation provider. Typically, it will be the topmost and leftmost row or column.
- IGridItemProvider does not provide for active manipulation of the grid such as merging or splitting cells.
- Controls that implement IGridItemProvider can typically be traversed (that is, a UI Automation client can move to adjacent controls) by using the keyboard.

Required Members for IGridItemProvider

The following properties and methods are required for implementing IGridItemProvider.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
Row	Property	None
Column	Property	None
RowSpan	Property	None
ColumnSpan	Property	None
ContainingGrid	Property	None

This control pattern has no associated methods or events.

Exceptions

This control pattern has no associated exceptions.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Implementing the UI Automation Grid Control Pattern
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Invoke Control Pattern

11/23/2019 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IInvokeProvider, including information about events and properties. Links to additional references are listed at the end of the topic.

The InvokePattern control pattern is used to support controls that do not maintain state when activated but rather initiate or perform a single, unambiguous action. Controls that do maintain state, such as check boxes and radio buttons, must instead implement IToggleProvider and ISelectionItemProvider respectively. For examples of controls that implement the Invoke control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Invoke control pattern, note the following guidelines and conventions:

- Controls implement IInvokeProvider if the same behavior is not exposed through another control pattern provider. For example, if the Invoke method on a control performs the same action as the Expand or Collapse method, the control should not implement IInvokeProvider.
- Invoking a control is generally performed by clicking or double-clicking or pressing ENTER, a predefined keyboard shortcut, or some alternate combination of keystrokes.
- InvokedEvent is raised on a control that has been activated (as a response to a control carrying out its associated action). If possible, the event should be raised after the control has completed the action and returned without blocking. The Invoked event should be raised before servicing the Invoke request in the following scenarios:
  - It is not possible or practical to wait until the action is complete.
  - The action requires user interaction.
  - The action is time-consuming and will cause the calling client to block for a significant amount of time.
- If invoking the control has significant side-effects, those side-effects should be exposed through the HelpText property. For example, even though Invoke is not associated with selection, Invoke may cause another control to become selected.
- Hover (or mouse-over) effects generally do not constitute an Invoked event. However, controls that perform an action (as opposed to cause a visual effect) based on the hover state should support the InvokePattern control pattern.

NOTE

This implementation is considered an accessibility issue if the control can be invoked only as a result of a mouse-related side effect.

- Invoking a control is different from selecting an item. However, depending on the control, invoking it may cause the item to become selected as a side-effect. For example, invoking a Microsoft Word document list item in the My Documents folder both selects the item and opens the document.
- An element can disappear from the UI Automation tree immediately upon being invoked. Requesting information from the element provided by the event callback may fail as a result. Pre-fetching cached information is the recommended workaround.
- Controls can implement multiple control patterns. For example, the Fill Color control on the Microsoft Excel toolbar implements both the InvokePattern and the ExpandCollapsePattern control patterns. ExpandCollapsePattern exposes the menu and the InvokePattern fills the active selection with the chosen color.

Required Members for IInvokeProvider

The following properties and methods are required for implementing IInvokeProvider.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
Invoke	method	Invoke is an asynchronous call and must return immediately without blocking. This behavior is particularly critical for controls that, directly or indirectly, launch a modal dialog when invoked. Any UI Automation client that instigated the event will remain blocked until the modal dialog is closed.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
ElementNotEnabledException	If the control is not enabled.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Invoke a Control Using UI Automation
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation MultipleView Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IMultipleViewProvider, including information about events and properties. Links to additional references are listed at the end of the topic.

The MultipleViewPattern control pattern is used to support controls that provide, and are able to switch between, multiple representations of the same set of information or child controls.

Examples of controls that can present multiple views include the list view (which can show its contents as thumbnails, tiles, icons, or details), Microsoft Excel charts (pie, line, bar, cell value with a formula), Microsoft Word documents (normal, Web layout, print layout, reading layout, outline), Microsoft Outlook calendar (year, month, week, day), and Microsoft Windows Media Player skins. The supported views are determined by the control developer and are specific to each control.

Implementation Guidelines and Conventions

When implementing the Multiple View control pattern, note the following guidelines and conventions:

- IMultipleViewProvider should also be implemented on a container that manages the current view if it is different from a control that provides the current view. For example, Windows Explorer contains a List control for the current folder content while the view for the control is managed from the Windows Explorer application.
- A control that is able to sort its content is not considered to support multiple views.
- The collection of views must be identical across instances.
- View names must be suitable for use in Text to Speech, Braille, and other human-readable applications.

Required Members for IMultipleViewProvider

The following properties and methods are required for implementing IMultipleViewProvider.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
CurrentView	Property	None
GetSupportedViews	Method	None
GetViewName	Method	None
SetCurrentView	Method	None

There are no events associated with this control pattern.

Exceptions

Provider must throw the following exceptions.

EXCEPTION TYPE	CONDITION
ArgumentException	When either SetCurrentView or GetViewName is called with a parameter that is not a member of the supported views collection.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation RangeValue Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

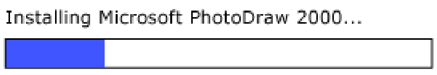
This topic introduces guidelines and conventions for implementing IRangeValueProvider, including information about events and properties. Links to additional references are listed at the end of the topic.

The RangeValuePattern control pattern is used to support controls that can be set to a value within a range. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Range Value control pattern, note the following guidelines and conventions:

- Controls allow recalibration of their supported properties based upon locale or user preference. An example of this is a thermometer control that can be set to display the temperature in Fahrenheit or Celsius.
- Controls that have ambiguous range values, such as progress bars or sliders, should have those values normalized.



Example of a Progress Bar Where Value Is of Type Integer and Minimum and Maximum Property Values Are Normalized to 0 and 100, Respectively

Required Members for IRangeValueProvider

REQUIRED MEMBER	MEMBER TYPE	NOTES
IsReadOnlyProperty	Property	None
ValueProperty	Property	None
LargeChangeProperty	Property	None
SmallChangeProperty	Property	None
MaximumProperty	Property	None
MinimumProperty	Property	None
SetValue	Methods	None

This control pattern has no associated events.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
ArgumentOutOfRangeException	SetValue is called with a value that is either greater than MaximumProperty or less than MinimumProperty.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Scroll Control Pattern

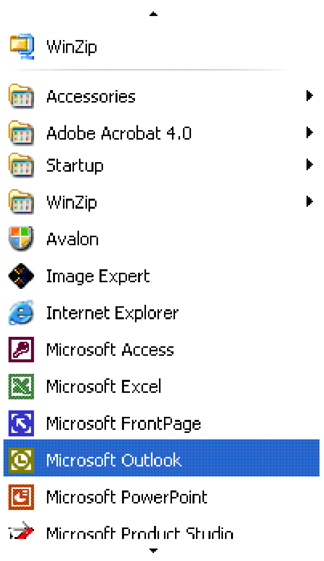
3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IScrollProvider, including information about events and properties. Links to additional references are listed at the end of the topic.

The ScrollPattern control pattern is used to support a control that acts as a scrollable container for a collection of child objects. The control is not required to use scrollbars to support the scrolling functionality, although it commonly does.



Example of a Scrolling Control that Does Not Use Scrollbars

For examples of controls that implement this control, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Scroll control pattern, note the following guidelines and conventions:

- The children of this control must implement IScrollItemProvider.
- The scrollbars of a container control do not support the ScrollPattern control pattern. They must support the RangeValuePattern control pattern instead.
- When scrolling is measured in percentages, all values or amounts related to scroll graduation must be normalized to a range of 0 to 100.
- HorizontallyScrollableProperty and VerticallyScrollableProperty are independent of the IsEnabledProperty.
- If HorizontallyScrollableProperty = `false` then HorizontalViewSizeProperty should be set to 100% and HorizontalScrollPercentProperty should be set to NoScroll. Likewise, if VerticallyScrollableProperty = `false`

then VerticalViewSizeProperty should be set to 100 percent and VerticalScrollPercentProperty should be set to NoScroll. This allows a UI Automation client to use these property values within the SetScrollPercent method while avoiding a race condition if a direction the client is not interested in scrolling becomes activated.

- HorizontalScrollPercent is locale-specific. Setting HorizontalScrollPercent = 100.0 must set the scrolling location of the control to the equivalent of its rightmost position for languages such as English that read left to right. Alternately, for languages such as Arabic that read right to left, setting HorizontalScrollPercent = 100.0 must set the scroll location to the leftmost position.

Required Members for IScrollProvider

The following properties and methods are required for implementing IScrollProvider.

REQUIRED MEMBER	MEMBER TYPE	NOTES
HorizontalScrollPercent	Property	None
VerticalScrollPercent	Property	None
HorizontalViewSize	Property	None
VerticalViewSize	Property	None
HorizontallyScrollable	Property	None
VerticallyScrollable	Property	None
Scroll	Method	None
SetScrollPercent	Method	None

This control pattern has no associated events.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
ArgumentException	Scroll throws this exception if a control supports SmallIncrement values exclusively for horizontal or vertical scrolling, but a LargeIncrement value is passed in.
ArgumentException	SetScrollPercent throws this exception when a value that cannot be converted to a double is passed in.
ArgumentOutOfRangeException	SetScrollPercent throws this exception when a value greater than 100 or less than 0 is passed in (except -1 which is equivalent to NoScroll).
InvalidOperationException	Both Scroll and SetScrollPercent throw this exception when an attempt is made to scroll in an unsupported direction.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation ScrollItem Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing the IScrollItemProvider, including information about properties, methods, and events. Links to additional references are listed at the end of the topic.

The ScrollItemPattern control pattern is used to support individual child controls of containers that implement IScrollProvider. This control pattern acts as a communication channel between a child control and its container to ensure that the container can change the currently visible content (or region) within its viewport to display the child control. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Scroll Item control pattern, note the following guidelines and conventions:

- Items contained within a Window or Canvas control are not required to implement the IScrollItemProvider interface. As an alternative, however, they must expose a valid location for the BoundingRectangleProperty. This will allow a UI Automation client application to use the ScrollPattern control pattern methods on the container to display the child item.

Required Members for IScrollItemProvider

The following method is required for implementing the IScrollProvider interface.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
ScrollIntoView	- Method	None

This control pattern has no associated properties or events.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
InvalidOperationException	If an item cannot be scrolled into view: - ScrollIntoView

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Selection Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

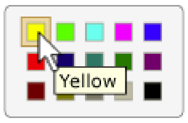
This topic introduces guidelines and conventions for implementing ISelectionProvider, including information about events and properties. Links to additional references are listed at the end of the topic.

The SelectionPattern control pattern is used to support controls that act as containers for a collection of selectable child items. The children of this element must implement ISelectionItemProvider. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Selection control pattern, note the following guidelines and conventions:

- Controls that implement ISelectionProvider allow either single or multiple child items to be selected. For example, list box, list view, and tree view support multiple selections whereas combo box, slider, and radio button group support single selection.
- Controls that have a minimum, maximum, and continuous range, such as the **Volume** slider control, should implement IRangeValueProvider instead of ISelectionProvider.
- Single-selection controls that manage child controls that implement IRawElementProviderFragmentRoot, such as the **Screen Resolution** slider in the **Display Properties** dialog box or the **Color Picker** selection control from Microsoft Word (illustrated below), should implement ISelectionProvider; their children should implement both IRawElementProviderFragment and ISelectionItemProvider.



Example of Color Swatch String Mapping

- Menus do not support SelectionPattern. If you are working with menu items that include both graphics and text (such as the **Preview Pane** items in the **View** menu in Microsoft Outlook) and need to convey state, you should implement IToggleProvider.

Required Members for ISelectionProvider

The following properties, methods, and events are required for the ISelectionProvider interface.

REQUIRED MEMBERS	TYPE	NOTES

REQUIRED MEMBERS	TYPE	NOTES
CanSelectMultiple	Property	Should support property changed events using AddAutomationPropertyChangedEventHandler and RemoveAutomationPropertyChangedEventHandler.
IsSelectionRequired	Property	Should support property changed events using AddAutomationPropertyChangedEventHandler and RemoveAutomationPropertyChangedEventHandler.
GetSelection	Method	None
InvalidatedEvent	Event	Raised when a selection in a container has changed significantly and requires sending more addition and removal events than the InvalidateLimit constant permits.

The IsSelectionRequired and CanSelectMultiple properties can be dynamic. For example, the initial state of a control might not have any items selected by default, indicating that IsSelectionRequired is `false`. However, after an item is selected, the control must always have at least one item selected. Similarly, in rare cases, a control might allow multiple items to be selected on initialization, but subsequently allow only single selections to be made.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
ElementNotEnabledException	If the control is not enabled.
InvalidOperationException	If the control is hidden.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Implementing the UI Automation SelectionItem Control Pattern
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation SelectionItem Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing ISelectionItemProvider, including information about properties, methods, and events. Links to additional references are listed at the end of the overview.

The SelectionItemPattern control pattern is used to support controls that act as individual, selectable child items of container controls that implement ISelectionProvider. For examples of controls that implement the SelectionItem control pattern, see Control Pattern Mapping for UI Automation Clients

Implementation Guidelines and Conventions

When implementing the Selection Item control pattern, note the following guidelines and conventions:

- Single-selection controls that manage child controls that implement IRawElementProviderFragmentRoot, such as the **Screen Resolution** slider in the **Display Properties** dialog box, should implement ISelectionProvider and their children should implement both IRawElementProviderFragment and ISelectionItemProvider.

Required Members for ISelectionItemProvider

The following properties, methods, and events are required for implementing ISelectionItemProvider.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
CanSelectMultiple	Property	None
IsSelectionRequired	Property	None
GetSelection	Method	None
InvalidatedEvent	Event	Raised when a selection in a container has changed significantly and requires sending more ElementSelectedEvent and ElementRemovedFromSelectionEvent events than the InvalidateLimit constant permits.

- If the result of a Select, an AddToSelection, or a RemoveFromSelection is a single selected item, an ElementSelectedEvent should be raised; otherwise send ElementAddedToSelectionEvent/ ElementRemovedFromSelectionEvent as appropriate.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
InvalidOperationException	When any of the following are attempted: - RemoveFromSelection is called on a single-selection container where IsSelectionRequiredProperty = `true` and an element is already selected. - RemoveFromSelection is called on a multiple-selection container where IsSelectionRequiredProperty = `true` and only one element is selected. - AddToSelection is called on a single-selection container where CanSelectMultipleProperty = `false` and another element is already selected.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Implementing the UI Automation Selection Control Pattern
- UI Automation Tree Overview
- Use Caching in UI Automation
- Fragment Provider Sample

Implementing the UI Automation Table Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing ITableProvider, including information about properties, methods, and events. Links to additional references are listed at the end of the overview.

The TablePattern control pattern is used to support controls that act as containers for a collection of child elements. The children of this element must implement ITableItemProvider and be organized in a two-dimensional logical coordinate system that can be traversed by row and column. This control pattern is analogous to IGridProvider, with the distinction that any control implementing ITableProvider must also expose a column and/or row header relationship for each child element. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

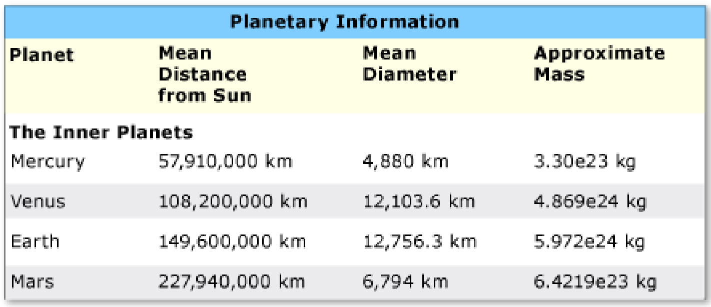
When implementing the Table control pattern, note the following guidelines and conventions:

- Access to the content of individual cells is through a two-dimensional logical coordinate system or array provided by the required concurrent implementation of IGridProvider.
- A column or row header can be contained within a table object or be a separate header object that is associated with a table object.
- Column and row headers may include both a primary header as well as any supporting headers.

NOTE

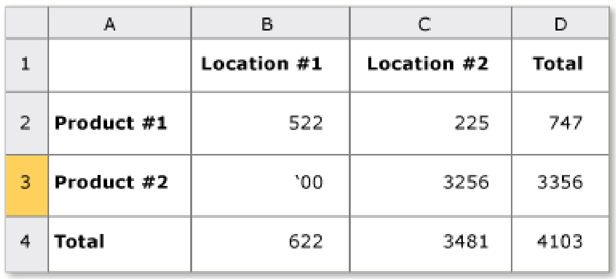
This concept becomes evident in a Microsoft Excel spreadsheet where a user has defined a "First name" column. This column now has two headers—the "First name" header defined by the user and the alphanumeric designation for that column assigned by the application.

- See Implementing the UI Automation Grid Control Pattern for related grid functionality.



Planetary Information			
Planet	**Mean Distance from Sun**	**Mean Diameter**	**Approximate Mass**
The Inner Planets			
Mercury	57,910,000 km	4,880 km	3.30e23 kg
Venus	108,200,000 km	12,103.6 km	4.869e24 kg
Earth	149,600,000 km	12,756.3 km	5.972e24 kg
Mars	227,940,000 km	6,794 km	6.4219e23 kg

Example of a Table with Complex Column Headers



	A	B	C	D
1		Location #1	Location #2	Total
2	Product #1	522	225	747
3	Product #2	`00	3256	3356
4	Total	622	3481	4103

Example of a Table with Ambiguous RowOrColumnMajor Property

Required Members for ITableProvider

The following properties and methods are required for the ITableProvider interface.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
RowOrColumnMajor	Property	None
GetColumnHeaders	Method	None
GetRowHeaders	Method	None

This control pattern has no associated events.

Exceptions

This control pattern has no associated exceptions.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Implementing the UI Automation TableItem Control Pattern
- Implementing the UI Automation Grid Control Pattern
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation TableItem Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing ITableItemProvider, including information about events and properties. Links to additional references are listed at the end of the overview.

The TableItemPattern control pattern is used to support child controls of containers that implement ITableProvider. Access to individual cell functionality is provided by the required concurrent implementation of IGridItemProvider. This control pattern is analogous to IGridItemProvider with the distinction that any control implementing ITableItemProvider must programmatically expose the relationship between the individual cell and its row and column information. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

- For related grid item functionality, see Implementing the UI Automation GridItem Control Pattern.

Required Members for ITableItemProvider

REQUIRED MEMBER	MEMBER TYPE	NOTES
GetColumnHeaderItems	Method	None
GetRowHeaderItems	Method	None

This control pattern has no associated properties or events.

Exceptions

This control pattern has no associated exceptions.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Implementing the UI Automation Table Control Pattern
- Implementing the UI Automation GridItem Control Pattern
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Toggle Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IToggleProvider, including information about methods and properties. Links to additional references are listed at the end of the topic.

The TogglePattern control pattern is used to support controls that can cycle through a set of states and maintain a state once set. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Toggle control pattern, note the following guidelines and conventions:

- Controls that do not maintain state when activated, such as buttons, toolbar buttons, and hyperlinks, must implement IInvokeProvider instead.
- A control must cycle through its ToggleState in the following order: On, Off and, if supported, Indeterminate.
- TogglePattern does not provide a SetState(newState) method due to issues surrounding the direct setting of a tri-state CheckBox without cycling through its appropriate ToggleState sequence.
- The RadioButton control does not implement IToggleProvider, as it is not capable of cycling through its valid states.

Required Members for IToggleProvider

The following properties and methods are required for implementing IToggleProvider.

REQUIRED MEMBER	MEMBER TYPE	NOTES
Toggle	Method	None
ToggleStateProperty	Property	None

This control pattern has no associated events.

Exceptions

This control pattern has no associated exceptions.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- Get the Toggle State of a Check Box Using UI Automation
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Transform Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE
This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing ITransformProvider, including information about properties, methods, and events. Links to additional references are listed at the end of the topic.

The TransformPattern control pattern is used to support controls that can be moved, resized, or rotated within a two-dimensional space. For examples of controls that implement this control pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Transform control pattern, note the following guidelines and conventions:

- Support for this control pattern is not limited to objects on the desktop. This control pattern must also be supported by the children of a container object if the children can be moved, resized, or rotated freely within the boundaries of the container.
- An object cannot be moved, resized, or rotated such that its resulting screen location would be completely outside the coordinates of its container and therefore inaccessible to the keyboard or mouse (for example, when a top-level window is moved off-screen or a child object is moved outside the boundaries of the container's viewport). In these cases, the object is placed as close to the requested screen coordinates as possible with the top or left coordinates overridden to be within the container boundaries.
- For multi-monitor systems, if an object is moved, resized, or rotated completely outside the combined desktop screen coordinates, the object is placed on the primary monitor as close to the requested coordinates as possible.
- All parameters and property values are absolute and independent of locale.

Required Members for ITransformProvider

The following properties and methods are required for implementing ITransformProvider.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
CanMove	Property	None
CanResize	Property	None
CanRotate	Property	None
Move	Method	None

REQUIRED MEMBERS	MEMBER TYPE	NOTES
Resize	Method	None
Rotate	Method	None

This control pattern has no associated events.

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
InvalidOperationException	Move - If the CanMoveProperty is false.
InvalidOperationException	Resize - If the CanResizeProperty is false.
InvalidOperationException	Rotate - If the CanRotateProperty is false.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Value Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

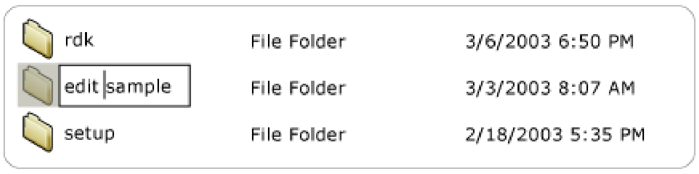
This topic introduces guidelines and conventions for implementing IValueProvider, including information on events and properties. Links to additional references are listed at the end of the topic.

The ValuePattern control pattern is used to support controls that have an intrinsic value not spanning a range and that can be represented as a string. This string can be editable, depending on the control and its settings. For examples of controls that implement this pattern, see Control Pattern Mapping for UI Automation Clients.

Implementation Guidelines and Conventions

When implementing the Value control pattern, note the following guidelines and conventions:

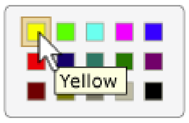
- Controls such as ListItem and TreeItem must support ValuePattern if the value of any of the items is editable, regardless of the current edit mode of the control. The parent control must also support ValuePattern if the child items are editable.



rdk	File Folder	3/6/2003 6:50 PM
edit sample	File Folder	3/3/2003 8:07 AM
setup	File Folder	2/18/2003 5:35 PM

Example of an Editable List Item

- Single-line edit controls support programmatic access to their contents by implementing IValueProvider. However, multi-line edit controls do not implement IValueProvider; instead they provide access to their content by implementing ITextProvider.
- To retrieve the textual contents of a multi-line edit control, the control must implement ITextProvider. However, ITextProvider does not support setting the value of a control.
- IValueProvider does not support the retrieval of formatting information or substring values. Implement ITextProvider in these scenarios.
- IValueProvider must be implemented by controls such as the **Color Picker** selection control from Microsoft Word (illustrated below), which supports string mapping between a color value (for example, "yellow") and an equivalent internal RGB structure.



Example of Color Swatch String Mapping

- A control should have its IsEnabledProperty set to `true` and its IsReadOnlyProperty set to `false` before allowing a call to SetValue.

Required Members for IValueProvider

The following properties and methods are required for implementing IValueProvider.

REQUIRED MEMBERS	MEMBER TYPE	NOTES
IsReadOnlyProperty	Property	None
ValueProperty	Property	None
SetValue	Method	None

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
InvalidOperationException	SetValue - If locale-specific information is passed to a control in an incorrect format such as an incorrectly formatted date.
ArgumentException	SetValue - If a new value cannot be converted from a string to a format the control recognizes.
ElementNotEnabledException	SetValue - When an attempt is made to manipulate a control that is not enabled.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- ValuePattern Insert Text Sample
- UI Automation Tree Overview
- Use Caching in UI Automation

Implementing the UI Automation Window Control Pattern

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic introduces guidelines and conventions for implementing IWindowProvider, including information about WindowPattern properties, methods, and events. Links to additional references are listed at the end of the topic.

The WindowPattern control pattern is used to support controls that provide fundamental window-based functionality within a traditional graphical user interface (GUI). Examples of controls that must implement this control pattern include top-level application windows, multiple-document interface (MDI) child windows, resizable split pane controls, modal dialogs and balloon help windows.

Implementation Guidelines and Conventions

When implementing the Window control pattern, note the following guidelines and conventions:

- To support the ability to modify both window size and screen position using UI Automation, a control must implement ITransformProvider in addition to IWindowProvider.
- Controls that contain title bars and title bar elements that enable the control to be moved, resized, maximized, minimized, or closed are typically required to implement IWindowProvider.
- Controls such as tooltip pop-ups and combo box or menu drop-downs do not typically implement IWindowProvider.
- Balloon help windows are differentiated from basic tooltip pop-ups by the provision of a window-like Close button.
- Full-screen mode is not supported by IWindowProvider as it is feature-specific to an application and is not typical window behavior.

Required Members for IWindowProvider

The following properties, methods, and events are required for the IWindowProvider interface.

REQUIRED MEMBER	MEMBER TYPE	NOTES
InteractionState	Property	None
IsModal	Property	None
IsTopmost	Property	None
Maximizable	Property	None

REQUIRED MEMBER	MEMBER TYPE	NOTES
Minimizable	Property	None
VisualState	Property	None
Close	Method	None
SetVisualState	Method	None
WaitForInputIdle	Method	None
WindowClosedEvent	Event	None
WindowOpenedEvent	Event	None
WindowInteractionState	Event	Is not guaranteed to be ReadyForUserInteraction

Exceptions

Providers must throw the following exceptions.

EXCEPTION TYPE	CONDITION
InvalidOperationException	SetVisualState - When a control does not support a requested behavior.
ArgumentOutOfRangeException	WaitForInputIdle - When the parameter is not a valid number.

See also

- UI Automation Control Patterns Overview
- Support Control Patterns in a UI Automation Provider
- UI Automation Control Patterns for Clients
- UI Automation Tree Overview
- Use Caching in UI Automation

UI Automation Control Patterns How-to Topics

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section contains detailed information about implementing control patterns.

In This Section

Expose the Content of a Table Using UI Automation

11/23/2019 • 8 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how Microsoft UI Automation can be used to expose the content and intrinsic properties of each cell within a tabular control.

Example

The following code example demonstrates how to obtain a AutomationElement that represents the content of a table cell; cell properties such as row and column indices, row and column spans, and row and column header information are also obtained. This example uses a focus change event handler to simulate keyboard traversal of a tabular control that implements UI Automation. Information for each table item is exposed on a focus change event.

NOTE

Since focus changes are global desktop events, focus change events outside the table should be filtered. See the TrackFocus Sample for a related implementation.

```
/// --------------------------------------------------------------------
/// <summary>
/// Starts the target application and returns the AutomationElement
/// obtained from the targets window handle.
/// </summary>
/// <param name="exe">
/// The target application.
/// </param>
/// <param name="filename">
/// The text file to be opened in the target application
/// </param>
/// <returns>
/// An AutomationElement representing the target application.
/// </returns>
/// --------------------------------------------------------------------
private AutomationElement StartTarget(string exe, string filename)
{
    // Start text editor and load with a text file.
    Process p = Process.Start(exe, filename);

    // targetApp --> the root AutomationElement
    AutomationElement targetApp =
        AutomationElement.FromHandle(p.MainWindowHandle);

    return targetApp;
}
```

```
''' --------------------------------------------------------------------
''' <summary>
''' Starts the target application and returns the AutomationElement
''' obtained from the targets window handle.
''' </summary>
''' <param name="exe">
''' The target application.
''' </param>
''' <param name="filename">
''' The text file to be opened in the target application
''' </param>
''' <returns>
''' An AutomationElement representing the target application.
''' </returns>
''' --------------------------------------------------------------------
Private Function StartTarget(ByVal exe As String, ByVal filename As String) As AutomationElement
    ' Start text editor and load with a text file.
    Dim p As Process = Process.Start(exe, filename)

    ' targetApp --> the root AutomationElement
    Dim targetApp As AutomationElement = _
    AutomationElement.FromHandle(p.MainWindowHandle)

    Return targetApp
End Function
```

```
/// --------------------------------------------------------------------
/// <summary>
/// Obtain the table control of interest from the target application.
/// </summary>
/// <param name="targetApp">
/// The target application.
/// </param>
/// <returns>
/// An AutomationElement representing a table control.
/// </returns>
/// --------------------------------------------------------------------
private AutomationElement GetTableElement(AutomationElement targetApp)
{
    // The control type we're looking for; in this case 'Document'
    PropertyCondition cond1 =
        new PropertyCondition(
        AutomationElement.ControlTypeProperty,
        ControlType.Table);

    // The control pattern of interest; in this case 'TextPattern'.
    PropertyCondition cond2 =
        new PropertyCondition(
        AutomationElement.IsTablePatternAvailableProperty,
        true);

    AndCondition tableCondition = new AndCondition(cond1, cond2);

    AutomationElement targetTableElement =
        targetApp.FindFirst(TreeScope.Descendants, tableCondition);

    // If targetTableElement is null then a suitable table control
    // was not found.
    return targetTableElement;
}
```

```vb
''' --------------------------------------------------------------------
''' <summary>
''' Obtain the table control of interest from the target application.
''' </summary>
''' <param name="targetApp">
''' The target application.
''' </param>
''' <returns>
''' An AutomationElement representing a table control.
''' </returns>
''' --------------------------------------------------------------------
Private Function GetTableElement(ByVal targetApp As AutomationElement) As AutomationElement
    ' The control type we're looking for; in this case 'Document'
    Dim cond1 As PropertyCondition = _
        New PropertyCondition( _
        AutomationElement.ControlTypeProperty, _
        ControlType.Table)

    ' The control pattern of interest; in this case 'TextPattern'.
    Dim cond2 As PropertyCondition = _
        New PropertyCondition( _
        AutomationElement.IsTablePatternAvailableProperty, _
        True)

    Dim tableCondition As AndCondition = New AndCondition(cond1, cond2)

    Dim targetTableElement As AutomationElement = _
    targetApp.FindFirst(TreeScope.Descendants, tableCondition)

    ' If targetTableElement is null then a suitable table control
    ' was not found.
    Return targetTableElement
End Function
```

```
///--------------------------------------------------------------------
/// <summary>
/// Obtains a TableItemPattern control pattern from an
/// AutomationElement.
/// </summary>
/// <param name="targetControl">
/// The AutomationElement of interest.
/// </param>
/// <returns>
/// A TableItemPattern object.
/// </returns>
///--------------------------------------------------------------------
private TableItemPattern GetTableItemPattern(
    AutomationElement targetControl)
{
    TableItemPattern tableItemPattern = null;

    try
    {
        tableItemPattern =
            targetControl.GetCurrentPattern(
            TableItemPattern.Pattern)
            as TableItemPattern;
    }
    // Object doesn't support the
    // TableItemPattern control pattern
    catch (InvalidOperationException)
    {
        return null;
    }

    return tableItemPattern;
}
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Obtains a TableItemPattern control pattern from an
''' AutomationElement.
''' </summary>
''' <param name="targetControl">
''' The AutomationElement of interest.
''' </param>
''' <returns>
''' A TableItemPattern object.
''' </returns>
'''--------------------------------------------------------------------
Private Function GetTableItemPattern( _
ByVal targetControl As AutomationElement) As TableItemPattern
    Dim tableItemPattern As TableItemPattern = Nothing

    Try
        tableItemPattern = DirectCast( _
        targetControl.GetCurrentPattern(tableItemPattern.Pattern), TableItemPattern)
    Catch exc As InvalidOperationException
        ' Object doesn't support the
        ' GridPattern control pattern
        Return Nothing
    End Try

    Return tableItemPattern

End Function 'GetTableItemPattern
```

```csharp
///--------------------------------------------------------------------
/// <summary>
/// Obtains a TablePattern control pattern from an
/// AutomationElement.
/// </summary>
/// <param name="targetControl">
/// The AutomationElement of interest.
/// </param>
/// <returns>
/// A TablePattern object.
/// </returns>
///--------------------------------------------------------------------
private TablePattern GetTablePattern(
    AutomationElement targetControl)
{
    TablePattern tablePattern = null;

    try
    {
        tablePattern =
            targetControl.GetCurrentPattern(
            TablePattern.Pattern)
            as TablePattern;
    }
    // Object doesn't support the
    // TablePattern control pattern
    catch (InvalidOperationException)
    {
        return null;
    }

    return tablePattern;
}
```

```vb
'''--------------------------------------------------------------------
''' <summary>
''' Obtains a TablePattern control pattern from an
''' AutomationElement.
''' </summary>
''' <param name="targetControl">
''' The AutomationElement of interest.
''' </param>
''' <returns>
''' A TablePattern object.
''' </returns>
'''--------------------------------------------------------------------
Private Function GetTablePattern( _
ByVal targetControl As AutomationElement) As TablePattern
    Dim tablePattern As TablePattern = Nothing

    Try
        tablePattern = DirectCast( _
        targetControl.GetCurrentPattern(tablePattern.Pattern), _
        TablePattern)
    Catch exc As InvalidOperationException
        ' Object doesn't support the
        ' TablePattern control pattern
        Return Nothing
    End Try

    Return tablePattern

End Function 'GetTablePattern
```

```
///--------------------------------------------------------------------
/// <summary>
/// Set up table item event listeners.
/// </summary>
/// <remarks>
/// The event listener is essentially a focus change listener.
/// Since this is a global desktop listener, a filter would be required
/// to ignore focus change events outside the table.
/// </remarks>
///--------------------------------------------------------------------
private void SetTableItemEventListeners()
{
    AutomationFocusChangedEventHandler tableItemFocusChangedListener =
        new AutomationFocusChangedEventHandler(OnTableItemFocusChange);
    Automation.AddAutomationFocusChangedEventHandler(
        tableItemFocusChangedListener);
}
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Set up table item event listeners.
''' </summary>
''' <remarks>
''' The event listener is essentially a focus change listener.
''' Since this is a global desktop listener, a filter would be required
''' to ignore focus change events outside the table.
''' </remarks>
'''--------------------------------------------------------------------
Private Sub SetTableItemEventListeners( _
ByVal targetControl As AutomationElement)
    Dim tableItemFocusChangedListener As AutomationFocusChangedEventHandler = _
    AddressOf OnTableItemFocusChange
    Automation.AddAutomationFocusChangedEventHandler( _
    tableItemFocusChangedListener)

End Sub
```

```
///--------------------------------------------------------------------
/// <summary>
/// Event handler for table item focus change.
/// Can be used to track traversal of individual table items
/// within a table.
/// </summary>
/// <param name="src">Object that raised the event.</param>
/// <param name="e">Event arguments.</param>
///--------------------------------------------------------------------
private void OnTableItemFocusChange(
    object src, AutomationFocusChangedEventArgs e)
{
    // Make sure the element still exists. Elements such as tooltips
    // can disappear before the event is processed.
    AutomationElement sourceElement;
    try
    {
        sourceElement = src as AutomationElement;
    }
    catch (ElementNotAvailableException)
    {
        return;
    }

    // Get a TableItemPattern from the source of the event.
    TableItemPattern tableItemPattern =
        GetTableItemPattern(sourceElement);
    if (tableItemPattern == null)
    {
        return;
    }

    // Get a TablePattern from the container of the current element.
    TablePattern tablePattern =
        GetTablePattern(tableItemPattern.Current.ContainingGrid);

    if (tablePattern == null)
    {
        return;
    }

    AutomationElement tableItem = null;
    try
    {
        tableItem = tablePattern.GetItem(
        tableItemPattern.Current.Row,
        tableItemPattern.Current.Column);
    }
    catch (ArgumentOutOfRangeException)
    {
        // If the requested row coordinate is larger than the RowCount
        // or the column coordinate is larger than the ColumnCount.
        // -- OR --
        // If either of the requested row or column coordinates
        // is less than zero.
        // TO DO: error handling.
    }

    // Further event processing can be done at this point.
    // For the purposes of this sample we can just record item properties.
    string controlType =
        tableItem.Current.ControlType.LocalizedControlType;
    AutomationElement[] columnHeaders =
        tableItemPattern.Current.GetColumnHeaderItems();
    AutomationElement[] rowHeaders =
        tableItemPattern.Current.GetRowHeaderItems();
    int itemRow = tableItemPattern.Current.Row;
    int itemColumn = tableItemPattern.Current.Column;
    int itemRowSpan = tableItemPattern.Current.RowSpan;
    int itemColumnSpan = tableItemPattern.Current.ColumnSpan;
}

///--------------------------------------------------------------------
/// <summary>
/// Handles the application shutdown.
/// </summary>
/// <param name="args">Event arguments.</param>
///--------------------------------------------------------------------
protected override void OnExit(System.Windows.ExitEventArgs args)
{
    Automation.RemoveAllEventHandlers();
    base.OnExit(args);
}
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Event handler for table item focus change.
''' Can be used to track traversal of individual table items
''' within a table.
''' </summary>
''' <param name="src">Object that raised the event.</param>
''' <param name="e">Event arguments.</param>
'''--------------------------------------------------------------------
Private Sub OnTableItemFocusChange(
Private Sub OnTableItemFocusChange( _
ByVal src As Object, ByVal e As AutomationFocusChangedEventArgs)
    ' Make sure the element still exists. Elements such as tooltips
    ' can disappear before the event is processed.
    Dim sourceElement As AutomationElement
    Try
        sourceElement = DirectCast(src, AutomationElement)
    Catch exc As ElementNotAvailableException
        Return
    End Try

    ' Get a TableItemPattern from the source of the event.
    Dim tableItemPattern As TableItemPattern = _
    GetTableItemPattern(sourceElement)

    If tableItemPattern Is Nothing Then
        Return
    End If

    ' Get a TablePattern from the container of the current element.
    Dim tablePattern As TablePattern = _
    GetTablePattern(tableItemPattern.Current.ContainingGrid)

    If tablePattern Is Nothing Then
        Return
    End If

    Dim tableItem As AutomationElement = Nothing
    Try
        tableItem = tablePattern.GetItem( _
        tableItemPattern.Current.Row, tableItemPattern.Current.Column)

    Catch exc As ArgumentOutOfRangeException
        ' If the requested row coordinate is larger than the RowCount
        ' or the column coordinate is larger than the ColumnCount.
        ' -- OR --
        ' If either of the requested row or column coordinates
        ' is less than zero.
        ' TO DO: error handling.
    End Try

    ' Further event processing can be done at this point.
    ' For the purposes of this sample we can just record item properties.
    Dim controlType As String = _
        tableItem.Current.ControlType.LocalizedControlType
    Dim columnHeaders As AutomationElement() = _
    tableItemPattern.Current.GetColumnHeaderItems()
    Dim rowHeaders As AutomationElement() = _
    tableItemPattern.Current.GetRowHeaderItems()
    Dim itemRow As Integer = tableItemPattern.Current.Row
    Dim itemColumn As Integer = tableItemPattern.Current.Column
    Dim itemRowSpan As Integer = tableItemPattern.Current.RowSpan
    Dim itemColumnSpan As Integer = tableItemPattern.Current.ColumnSpan

End Sub


'''--------------------------------------------------------------------
''' <summary>
''' Handles the application shutdown.
''' </summary>
''' <param name="args">Event arguments.</param>
'''--------------------------------------------------------------------
Protected Overrides Sub OnExit(ByVal args As System.Windows.ExitEventArgs)
    Automation.RemoveAllEventHandlers()
    MyBase.OnExit(args)

End Sub
```

See also

- UI Automation Control Patterns Overview
- UI Automation Control Patterns for Clients
- Implementing the UI Automation Table Control Pattern
- Implementing the UI Automation TableItem Control Pattern
- Implementing the UI Automation Grid Control Pattern
- Implementing the UI Automation GridItem Control Pattern

Get Supported UI Automation Control Patterns

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to retrieve control pattern objects from UI Automation elements.

Obtain All Control Patterns

1. Get the AutomationElement whose control patterns you are interested in.
2. Call GetSupportedPatterns to get all control patterns from the element.

Caution

It is strongly recommended that a client not use GetSupportedPatterns. Performance can be severely affected as this method calls GetCurrentPattern internally for each existing control pattern. If possible, a client should call GetCurrentPattern for the key patterns of interest.

Obtain a Specific Control Pattern

1. Get the AutomationElement whose control patterns you are interested in.
2. Call GetCurrentPattern or TryGetCurrentPattern to query for a specific pattern. These methods are similar, but if the pattern is not found, GetCurrentPattern raises an exception, and TryGetCurrentPattern returns `false`.

Example

The following example retrieves an AutomationElement for a list item and obtains a SelectionItemPattern from that element.

```csharp
/// <summary>
/// Sets the focus to a list and selects a string item in that list.
/// </summary>
/// <param name="listElement">The list element.</param>
/// <param name="itemText">The text to select.</param>
/// <remarks>
/// This deselects any currently selected items. To add the item to the current selection
/// in a multiselect list, use AddToSelection instead of Select.
/// </remarks>
public void SelectListItem(AutomationElement listElement, String itemText)
{
    if ((listElement == null) || (itemText == ""))
    {
        throw new ArgumentException("Argument cannot be null or empty.");
    }
    listElement.SetFocus();
    Condition cond = new PropertyCondition(
        AutomationElement.NameProperty, itemText, PropertyConditionFlags.IgnoreCase);
    AutomationElement elementItem = listElement.FindFirst(TreeScope.Children, cond);
    if (elementItem != null)
    {
        SelectionItemPattern pattern;
        try
        {
            pattern = elementItem.GetCurrentPattern(SelectionItemPattern.Pattern) as SelectionItemPattern;
        }
        catch (InvalidOperationException ex)
        {
            Console.WriteLine(ex.Message);  // Most likely "Pattern not supported."
            return;
        }
        pattern.Select();
    }
}
```

```vb
''' <summary>
''' Sets the focus to a list and selects a string item in that list.
''' </summary>
''' <param name="listElement">The list element.</param>
''' <param name="itemText">The text to select.</param>
''' <remarks>
''' This deselects any currently selected items. To add the item to the current selection
''' in a multiselect list, use AddToSelection instead of Select.
''' </remarks>
Public Sub SelectListItem(ByVal listElement As AutomationElement, ByVal itemText As String)
    If listElement Is Nothing OrElse itemText = "" Then
        Throw New ArgumentException("Argument cannot be null or empty.")
    End If
    listElement.SetFocus()
    Dim cond As New PropertyCondition(AutomationElement.NameProperty, itemText,
PropertyConditionFlags.IgnoreCase)
    Dim elementItem As AutomationElement = listElement.FindFirst(TreeScope.Children, cond)
    If Not (elementItem Is Nothing) Then
        Dim pattern As SelectionItemPattern
        Try
            pattern = DirectCast(elementItem.GetCurrentPattern(SelectionItemPattern.Pattern), _
                SelectionItemPattern)
        Catch ex As InvalidOperationException
            Console.WriteLine(ex.Message) ' Most likely "Pattern not supported."
            Return
        End Try
        pattern.Select()
    End If

End Sub
```

See also

- UI Automation Control Patterns for Clients

Get the Toggle State of a Check Box Using UI Automation

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to use Microsoft UI Automation to get the toggle state of a control.

Example

This example uses the GetCurrentPattern method of the AutomationElement class to obtain a TogglePattern object from a control and return its ToggleState property.

```
/// <summary>
/// Gets the toggle state of an element in the target application.
/// </summary>
/// <param name="element">The target element.</param>
private bool IsElementToggledOn(AutomationElement element)
{
    if (element == null)
    {
        // TODO: Invalid parameter error handling.
        return false;
    }

    Object objPattern;
    TogglePattern togPattern;
    if (true == element.TryGetCurrentPattern(TogglePattern.Pattern, out objPattern))
    {
        togPattern = objPattern as TogglePattern;
        return togPattern.Current.ToggleState == ToggleState.On;
    }
    // TODO: Object doesn't support TogglePattern error handling.
    return false;
}
```

```vb
''' <summary>
''' Gets the toggle state of an element in the target application.
''' </summary>
''' <param name="element">The target element.</param>
Private Function IsElementToggledOn(ByVal element As AutomationElement) As Boolean
    If element Is Nothing Then
        ' TODO: Invalid parameter error handling.
        Return False
    End If

    Dim objPattern As Object = Nothing
    Dim togPattern As TogglePattern
    If True = element.TryGetCurrentPattern(TogglePattern.Pattern, objPattern) Then
        togPattern = TryCast(objPattern, TogglePattern)
        Return togPattern.Current.ToggleState = ToggleState.On
    End If
    ' TODO: Object doesn't support TogglePattern error handling.
    Return False
End Function
```

Invoke a Control Using UI Automation

11/23/2019 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic demonstrates how to perform the following tasks:

- Find a control that matches specific property conditions by walking the control view of the UI Automation tree for the target application.
- Create an AutomationElement for each control.
- Obtain an InvokePattern object from any UI Automation element found that supports the InvokePattern control pattern.
- Use Invoke to invoke the control from a client event handler.

Example

This example uses the TryGetCurrentPattern method of the AutomationElement class to generate an InvokePattern object and invoke a control by using the Invoke method.

```
///--------------------------------------------------------------------
/// <summary>
/// Walks the UI Automation tree of the target and reports the control
/// type of each element it finds in the control view to the client.
/// </summary>
/// <param name="targetTreeViewElement">
/// The root of the search on this iteration.
/// </param>
/// <param name="elementIndex">
/// The TreeView index for this iteration.
/// </param>
/// <remarks>
/// This is a recursive function that maps out the structure of the
/// subtree of the target beginning at the AutomationElement passed in
/// as the rootElement on the first call. This could be, for example,
/// an application window.
/// CAUTION: Do not pass in AutomationElement.RootElement. Attempting
/// to map out the entire subtree of the desktop could take a very
/// long time and even lead to a stack overflow.
/// </remarks>
///--------------------------------------------------------------------
private void FindTreeViewDescendants(
    AutomationElement targetTreeViewElement, int treeviewIndex)
{
    if (targetTreeViewElement == null)
        return;

    AutomationElement elementNode =
        TreeWalker.ControlViewWalker.GetFirstChild(targetTreeViewElement);

    while (elementNode != null)
```

```
        {
            Label elementInfo = new Label();
            elementInfo.Margin = new Thickness(0);
            clientTreeViews[treeviewIndex].Children.Add(elementInfo);

            // Compile information about the control.
            elementInfoCompile = new StringBuilder();
            string controlName =
                (elementNode.Current.Name == "") ?
                "Unnamed control" : elementNode.Current.Name;
            string autoIdName =
                (elementNode.Current.AutomationId == "") ?
                "No AutomationID" : elementNode.Current.AutomationId;

            elementInfoCompile.Append(controlName)
                .Append(" (")
                .Append(elementNode.Current.ControlType.LocalizedControlType)
                .Append(" - ")
                .Append(autoIdName)
                .Append(")");

            // Test for the control patterns of interest for this sample.
            object objPattern;
            ExpandCollapsePattern expcolPattern;
            if (true == elementNode.TryGetCurrentPattern(ExpandCollapsePattern.Pattern, out objPattern))
            {
                expcolPattern = objPattern as ExpandCollapsePattern;
                if (expcolPattern.Current.ExpandCollapseState != ExpandCollapseState.LeafNode)
                {
                    Button expcolButton = new Button();
                    expcolButton.Margin = new Thickness(0, 0, 0, 5);
                    expcolButton.Height = 20;
                    expcolButton.Width = 100;
                    expcolButton.Content = "ExpandCollapse";
                    expcolButton.Tag = expcolPattern;
                    expcolButton.Click +=
                        new RoutedEventHandler(ExpandCollapse_Click);
                    clientTreeViews[treeviewIndex].Children.Add(expcolButton);
                }
            }
            TogglePattern togPattern;
            if (true == elementNode.TryGetCurrentPattern(TogglePattern.Pattern, out objPattern))
            {
                togPattern = objPattern as TogglePattern;
                Button togButton = new Button();
                togButton.Margin = new Thickness(0, 0, 0, 5);
                togButton.Height = 20;
                togButton.Width = 100;
                togButton.Content = "Toggle";
                togButton.Tag = togPattern;
                togButton.Click += new RoutedEventHandler(Toggle_Click);
                clientTreeViews[treeviewIndex].Children.Add(togButton);
            }
            InvokePattern invPattern;
            if (true == elementNode.TryGetCurrentPattern(InvokePattern.Pattern, out objPattern))
            {
                invPattern = objPattern as InvokePattern;
                Button invButton = new Button();
                invButton.Margin = new Thickness(0);
                invButton.Height = 20;
                invButton.Width = 100;
                invButton.Content = "Invoke";
                invButton.Tag = invPattern;
                invButton.Click += new RoutedEventHandler(Invoke_Click);
                clientTreeViews[treeviewIndex].Children.Add(invButton);
            }
            // Display compiled information about the control.
            elementInfo.Content = elementInfoCompile;
            Separator sep = new Separator();
            clientTreeViews[treeviewIndex].Children.Add(sep);

            // Iterate to next element.
            // elementNode - Current element.
            // treeviewIndex - Index of parent TreeView.
            FindTreeViewDescendants(elementNode, treeviewIndex);
            elementNode =
                TreeWalker.ControlViewWalker.GetNextSibling(elementNode);
        }
    }
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Walks the UI Automation tree of the target and reports the control
''' type of each element it finds in the control view to the client.
''' </summary>
''' <param name="targetTreeViewElement">
''' The root of the search on this iteration.
''' </param>
''' <param name="treeviewIndex">
''' The TreeView index for this iteration.
''' </param>
''' <remarks>
''' This is a recursive function that maps out the structure of the
''' subtree beginning at the AutomationElement passed in as
''' rootElement on the first call. This could be, for example,
''' an application window.
''' CAUTION: Do not pass in AutomationElement.RootElement. Attempting
''' to map out the entire subtree of the desktop could take a very
''' long time and even lead to a stack overflow.
''' </remarks>
'''--------------------------------------------------------------------
Private Sub FindTreeViewDescendants( _
ByVal targetTreeViewElement As AutomationElement, _
ByVal treeviewIndex As Integer)
    If (IsNothing(targetTreeViewElement)) Then
        Return
    End If

    Dim elementNode As AutomationElement = _
    TreeWalker.ControlViewWalker.GetFirstChild(targetTreeViewElement)

    While Not (elementNode Is Nothing)
        Dim elementInfo As New Label()
        elementInfo.Margin = New Thickness(0)
        clientTreeViews(treeviewIndex).Children.Add(elementInfo)

        ' Compile information about the control.
        elementInfoCompile = New StringBuilder()
        Dim controlName As String
        If (elementNode.Current.Name = "") Then
            controlName = "Unnamed control"
        Else
            controlName = elementNode.Current.Name
        End If
        Dim autoIdName As String
        If (elementNode.Current.AutomationId = "") Then
            autoIdName = "No AutomationID"
        Else
            autoIdName = elementNode.Current.AutomationId
        End If


        elementInfoCompile.Append(controlName).Append(" (") _
        .Append(elementNode.Current.ControlType.LocalizedControlType) _
        .Append(" - ").Append(autoIdName).Append(")")

        ' Test for the control patterns of interest for this sample.
        ' Test for the control patterns of interest for this sample.
        Dim objPattern As Object = Nothing
        Dim expcolPattern As ExpandCollapsePattern
        If True = elementNode.TryGetCurrentPattern(ExpandCollapsePattern.Pattern, objPattern) Then
            expcolPattern = DirectCast(objPattern, ExpandCollapsePattern)
            If expcolPattern.Current.ExpandCollapseState <> ExpandCollapseState.LeafNode Then
                Dim expcolButton As New Button()
                expcolButton.Margin = New Thickness(0, 0, 0, 5)
                expcolButton.Height = 20
                expcolButton.Width = 100
                expcolButton.Content = "ExpandCollapse"
                expcolButton.Tag = expcolPattern
                AddHandler expcolButton.Click, AddressOf ExpandCollapse_Click
                clientTreeViews(treeviewIndex).Children.Add(expcolButton)
            End If
        End If
        Dim togPattern As TogglePattern
        If True = elementNode.TryGetCurrentPattern(TogglePattern.Pattern, objPattern) Then
            togPattern = DirectCast(objPattern, TogglePattern)
            Dim togButton As New Button()
            togButton.Margin = New Thickness(0, 0, 0, 5)
            togButton.Height = 20
            togButton.Width = 100
            togButton.Content = "Toggle"
            togButton.Tag = togPattern
            AddHandler togButton.Click, AddressOf Toggle_Click
            clientTreeViews(treeviewIndex).Children.Add(togButton)
        End If
        Dim invPattern As InvokePattern
        If True = elementNode.TryGetCurrentPattern(InvokePattern.Pattern, objPattern) Then
            invPattern = DirectCast(objPattern, InvokePattern)
            Dim invButton As New Button()
            invButton.Margin = New Thickness(0)
            invButton.Height = 20
            invButton.Width = 100
            invButton.Content = "Invoke"
            invButton.Tag = invPattern
            AddHandler invButton.Click, AddressOf Invoke_Click
            clientTreeViews(treeviewIndex).Children.Add(invButton)
        End If
        ' Display compiled information about the control.
        elementInfo.Content = elementInfoCompile
        Dim sep As New Separator()
        clientTreeViews(treeviewIndex).Children.Add(sep)

        ' Iterate to next element.
        ' elementNode - Current element.
        ' treeviewIndex - Index of parent TreeView.
        FindTreeViewDescendants(elementNode, treeviewIndex)
        elementNode = TreeWalker.ControlViewWalker.GetNextSibling(elementNode)
    End While

End Sub
```

```
///--------------------------------------------------------------------
/// <summary>
/// Handles the Invoke click event on the client control.
/// The client click handler calls Invoke() on the equivalent target control.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
/// <remarks>
/// The Tag property of the FrameworkElement, the client button in this
/// case, is used to store the InvokePattern object previously obtained
/// from the associated target control.
/// </remarks>
///--------------------------------------------------------------------
void Invoke_Click(object sender, RoutedEventArgs e)
{
    Button clientButton = sender as Button;
    InvokePattern targetInvokePattern = clientButton.Tag as InvokePattern;
    if (targetInvokePattern == null)
        return;
    targetInvokePattern.Invoke();
    statusText.Text = "Button invoked.";
}
```

```
'''--------------------------------------------------------------------
''' <summary>
''' Handles the Invoke click event on the client control.
''' The client click handler calls Invoke() on the equivalent target control.
''' </summary>
''' <param name="sender">The object that raised the event.</param>
''' <param name="e">Event arguments.</param>
''' <remarks>
''' The Tag property of the FrameworkElement, the client button in this
''' case, is used to store the InvokePattern object previously obtained
''' from the associated target control.
''' </remarks>
'''--------------------------------------------------------------------
Private Sub Invoke_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim clientButton As Button = DirectCast(sender, Button)
    Dim targetInvokePattern As InvokePattern = _
    DirectCast(clientButton.Tag, InvokePattern)
    If (IsNothing(targetInvokePattern)) Then
        Return
    End If
    targetInvokePattern.Invoke()
    statusText.Text = "Button invoked."
End Sub
```

See also

- InvokePattern, ExpandCollapsePattern, and TogglePattern Sample

Move a UI Automation Element

1/8/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This example demonstrates how to move a UI Automation element to a specified screen location.

Example

The following example uses the WindowPattern and TransformPattern control patterns to programmatically move a Win32 target application to discrete screen locations and track the BoundingRectangleProperty AutomationPropertyChangedEvent.

```
/// <summary>
/// The Startup handler.
/// </summary>
/// <param name="e">The event arguments</param>
protected override void OnStartup(StartupEventArgs e)
{
    // Start the WindowMove client.
    CreateWindow();

    try
    {
        // Obtain an AutomationElement from the target window handle.
        targetWindow = StartTargetApp(targetApplication);

        // Does the automation element exist?
        if (targetWindow == null)
        {
            Feedback("No target.");
            return;
        }
        Feedback("Found target.");

        // find current location of our window
        targetLocation = targetWindow.Current.BoundingRectangle.Location;

        // Obtain required control patterns from our automation element
        windowPattern = GetControlPattern(targetWindow,
            WindowPattern.Pattern) as WindowPattern;

        if (windowPattern == null) return;

        // Make sure our window is usable.
        // WaitForInputIdle will return before the specified time
        // if the window is ready.
        if (false == windowPattern.WaitForInputIdle(10000))
        {
            Feedback("Object not responding in a timely manner.");
            return;
        }
        Feedback("Window ready for user interaction");
        // Register for required events
        RegisterForEvents(
            targetWindow, WindowPattern.Pattern, TreeScope.Element);

        // Obtain required control patterns from our automation element
        transformPattern =
            GetControlPattern(targetWindow, TransformPattern.Pattern)
            as TransformPattern;

        if (transformPattern == null) return;

        // Is the TransformPattern object moveable?
        if (transformPattern.Current.CanMove)
        {
            // Enable our WindowMove fields
            xCoordinate.IsEnabled = true;
            yCoordinate.IsEnabled = true;
            moveTarget.IsEnabled = true;

            // Move element
            transformPattern.Move(0, 0);
        }
        else
        {
            Feedback("Wndow is not moveable.");
        }
    }
    catch (ElementNotAvailableException)
    {
        Feedback("Client window no longer available.");
        return;
    }
    catch (InvalidOperationException)
    {
        Feedback("Client window cannot be moved.");
        return;
    }
    catch (Exception exc)
    {
        Feedback(exc.ToString());
    }
}
```

```
'' <summary>
'' The Startup handler.
'' </summary>
'' <param name="e">The event arguments</param>
Protected Overrides Sub OnStartup(ByVal e As StartupEventArgs)

    ' Start the WindowMove client.
    CreateWindow()

    Try
        ' Obtain an AutomationElement from the target window handle.
        targetWindow = StartTargetApp(targetApplication)

        ' Does the automation element exist?
        If targetWindow Is Nothing Then
            Feedback("No target.")
            Return
        End If
        Feedback("Found target.")

        ' Obtain required control patterns from our automation element
        windowPattern = CType( _
        GetControlPattern(targetWindow, windowPattern.Pattern),  _
        WindowPattern)
        If (windowPattern Is Nothing) Then Return

        ' Make sure our window is usable.
        ' WaitForInputIdle will return before the specified time
        ' if the window is ready.
        If (False = windowPattern.WaitForInputIdle(10000)) Then
            Feedback("Object not responding in a timely manner.")
            Return
        End If
        Feedback("Window ready for user interaction")

        ' Register for required events
        RegisterForEvents( _
        targetWindow, windowPattern.Pattern, TreeScope.Element)

        ' find current location of our window
        targetLocation = _
        targetWindow.Current.BoundingRectangle.Location

        ' Obtain required control patterns from our automation element
        transformPattern = CType( _
        GetControlPattern(targetWindow, transformPattern.Pattern),  _
        TransformPattern)

        If (transformPattern Is Nothing) Then Return

        ' Is the TransformPattern object moveable?
        If transformPattern.Current.CanMove Then
            ' Enable our WindowMove fields
            xCoordinate.IsEnabled = True
            yCoordinate.IsEnabled = True
            moveTarget.IsEnabled = True

            ' Move element
            transformPattern.Move(0, 0)
        Else
            Feedback("Wndow is not moveable.")
        End If

    Catch exc As ElementNotAvailableException
        Feedback("Client window no longer available.")

    Catch exc As InvalidOperationException
        Feedback("Client window cannot be moved.")
        Return

    Catch exc As Exception
        Feedback(exc.ToString())

    End Try

End Sub
```

```
/// <summary>
/// Handles the 'Move' button invoked event.
/// By default, the Move method does not allow an object
/// to be moved completely off-screen.
/// </summary>
/// <param name="src">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
private void btnMove_Click(object src, RoutedEventArgs e)
{
    try
    {
        // If coordinate left blank, substitute 0
        if (xCoordinate.Text == "") xCoordinate.Text = "0";
        if (yCoordinate.Text == "") yCoordinate.Text = "0";

        // Reset background colours
        xCoordinate.Background = System.Windows.Media.Brushes.White;
        yCoordinate.Background = System.Windows.Media.Brushes.White;

        if (windowPattern.Current.WindowVisualState ==
            WindowVisualState.Minimized)
            windowPattern.SetWindowVisualState(WindowVisualState.Normal);

        double X = double.Parse(xCoordinate.Text);
        double Y = double.Parse(yCoordinate.Text);

        // Should validate the requested screen location
        if ((X < 0) ||
            (X >= (SystemParameters.WorkArea.Width -
            targetWindow.Current.BoundingRectangle.Width)))
        {
            Feedback("X-coordinate would place the window all or partially off-screen.");
            xCoordinate.Background = System.Windows.Media.Brushes.Yellow;
        }

        if ((Y < 0) ||
            (Y >= (SystemParameters.WorkArea.Height -
            targetWindow.Current.BoundingRectangle.Height)))
        {
            Feedback("Y-coordinate would place the window all or partially off-screen.");
            yCoordinate.Background = System.Windows.Media.Brushes.Yellow;
        }

        // transformPattern was obtained from the target window.
        transformPattern.Move(X, Y);
    }
    catch (ElementNotAvailableException)
    {
        Feedback("Client window no longer available.");
        return;
    }
    catch (InvalidOperationException)
    {
        Feedback("Client window cannot be moved.");
        return;
    }
}
```

```
'' <summary>
'' Handles the 'Move' button invoked event.
'' By default, the Move method does not allow an object
'' to be moved completely off-screen.
'' </summary>
'' <param name="src">The object that raised the event.</param>
'' <param name="e">Event arguments.</param>
Private Sub btnMove_Click( _
ByVal sender As Object, ByVal e As RoutedEventArgs)
    Try
        ' If coordinate left blank, substitute 0
        If (xCoordinate.Text = "") Then xCoordinate.Text = "0"
        If (yCoordinate.Text = "") Then yCoordinate.Text = "0"

        ' Reset background colours
        xCoordinate.Background = System.Windows.Media.Brushes.White
        yCoordinate.Background = System.Windows.Media.Brushes.White

        If (windowPattern.Current.WindowVisualState = WindowVisualState.Minimized) Then
            windowPattern.SetWindowVisualState(WindowVisualState.Normal)
        End If

        Dim X As Double = Double.Parse(xCoordinate.Text)
        Dim Y As Double = Double.Parse(yCoordinate.Text)

        ' Should validate the requested screen location.
        If (X >= (SystemParameters.WorkArea.Width - targetWindow.Current.BoundingRectangle.Width)) Then
            Feedback("X-coordinate would place the window all or partially off-screen.")
            xCoordinate.Background = System.Windows.Media.Brushes.Yellow
        End If

        If (Y >= (SystemParameters.WorkArea.Height - targetWindow.Current.BoundingRectangle.Height)) Then
            Feedback("Y-coordinate would place the window all or partially off-screen.")
            yCoordinate.Background = System.Windows.Media.Brushes.Yellow
        End If

        ' transformPattern was obtained from the target window.
        transformPattern.Move(X, Y)

    Catch exc As ElementNotAvailableException
        Feedback("Client window no longer available.")

    Catch exc As InvalidOperationException
        Feedback("Client window cannot be moved.")
        Return

    Catch exc As Exception
        Feedback(exc.ToString())

    End Try

End Sub
```

See also

- WindowPattern Sample

UI Automation Text Pattern

12/4/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains overviews and how-to topics to help you get started in programming for text elements in Microsoft UI Automation.

In This Section

UI Automation TextPattern Overview
TextPattern and Embedded Objects Overview
How-to Topics

Reference

System.Windows.Automation.Text

See also

- UI Automation Control Patterns
- UI Automation Control Patterns for Clients

UI Automation TextPattern Overview

1/8/2020 • 7 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This overview describes how to use Microsoft UI Automation to expose the textual content, including format and style attributes, of text controls in UI Automation-supported platforms. These controls include, but are not limited to, the Microsoft .NET Framework TextBox and RichTextBox as well as their Win32 equivalents.

Exposing the textual content of a control is accomplished through the use of the TextPattern control pattern, which represents the contents of a text container as a text stream. In turn, TextPattern requires the support of the TextPatternRange class to expose format and style attributes. TextPatternRange supports TextPattern by representing contiguous or multiple, disjoint text spans in a text container with a collection of Start and End endpoints. TextPatternRange supports functionality such as selection, comparison, retrieval and traversal.

NOTE

The TextPattern classes do not provide a means to insert or modify text. However, depending on the control, this may be accomplished by the UI Automation ValuePattern or through direct keyboard input. See the TextPattern Insert Text Sample for an example.

The functionality described in this overview is vital to assistive technology vendors and their end users. Assistive technologies can use UI Automation to gather complete text formatting information for the user and provide programmatic navigation and selection of text by TextUnit (character, word, line, or paragraph).

UI Automation TextPattern vs. Text Services Framework

Text Services Framework (TSF) is a simple and scalable system framework that enables natural language services and advanced text input on the desktop and within applications. In addition to providing interfaces for applications to expose their text store it also supports metadata for that text store.

However, TSF was designed for applications that need to inject input into context-aware scenarios whereas TextPattern is a read-only solution (with the limited workaround noted above) meant to provide optimized access to a text store for screen-readers and Braille devices.

In short, accessible technologies that require read-only access to a text store can use TextPattern, but will need the more complex functionality of TSF for context-aware input.

Control Types

Text

The Text control is the basic element representing a piece of text on the screen.

A standalone text control can be used as a label or static text on a form. Text controls can also be contained within the structure of a ListItem, TreeItem or DataItem.

NOTE

Text controls might not appear in the content view of the UI Automation tree (see UI Automation Tree Overview). This is because text controls are often displayed through the Name property of another control. For instance, the text that is used to label an Edit control is exposed through the Name property of the Edit control. Because the Edit control is in the content view of the UI Automation tree, it is not necessary for the text element itself to be in that view of the UI Automation tree. The only text that shows up in the content view is text that is not redundant information. This enables any assistive technology to quickly filter only on the pieces of information that their users need.

Edit

Edit controls enable a user to view and edit a single line of text.

NOTE

The single line of text may wrap in certain layout scenarios.

Document

Document controls let a user navigate and obtain information from multiple pages of text.

TextPattern Client APIs

`System.Windows.Automation.TextPattern Class`	The entry point for the Microsoft UI Automation text model. This class also contains the two TextPattern event listeners, TextSelectionChangedEvent and TextChangedEvent.
`System.Windows.Automation.Text.TextPatternRange Class`	The representation of a span of text within a text container that supports TextPattern. UI Automation clients should be careful about the current validity of a text range created using TextPatternRange. If the original text in the text control is completely replaced by new text, the current text range becomes invalid. However, the text range may still have some viability if only part of the original text is changed and the underlying text control is managing its text "pointer" with anchors (or endpoints) rather than with absolute character positioning. Clients can listen for a TextChangedEvent for notification of any changes to the textual content they are working with.
`System.Windows.Automation.AutomationTextAttribute Class`	Used to identify the formatting attributes of a text range.

TextPattern Provider APIs

UI elements or controls that support TextPattern by implementing the ITextProvider and ITextRangeProvider interfaces, either natively or through Microsoft UI Automation proxies, are capable of exposing detailed attribute information for any text they contain in addition to providing robust navigational capabilities.

A TextPattern provider does not have to support all text attributes if the control lacks support for any particular attributes.

A TextPattern provider must support the GetSelection and Select functions if the control supports text selection or placement of the text cursor (or system caret) within the text area. If the control does not support this functionality then it does not need to support either of these methods. However, the control must expose the type of text selection it supports by implementing the SupportedTextSelection property.

A TextPattern provider must always support the TextUnit constants Character and Document as well as any other TextUnit constants it is capable of supporting.

> **NOTE**
>
> The provider may skip support for a specific TextUnit by deferring to the next largest TextUnit supported in the following order: Character, Format, Word, Line, Paragraph, Page, and Document.

`ITextProvider Interface`	Exposes methods, properties and attributes that support TextPattern in client applications (see ITextProvider).
`ITextRangeProvider Interface`	Represents a span of text in a text provider (see ITextRangeProvider).
`System.Windows.Automation.TextPatternIdentifiers Class`	Contains values that are used as identifiers for text providers (see TextPatternIdentifiers).

Security

The UI Automation architecture was designed with security in mind (see UI Automation Security Overview). However, the TextPattern classes described in this overview require some specific security considerations.

- Microsoft UI Automation text providers supply read-only interfaces and do not provide the ability to change the existing text in a control.
- UI Automation clients can only use Microsoft UI Automation if they are fully "trusted". An example of this would be the protected Logon Desktop, where only known and trusted applications can run.
- Developers of UI Automation providers should be aware that all information they choose to expose in their controls through Microsoft UI Automation is essentially public and fully accessible by other code. Microsoft UI Automation makes no effort to determine the trustworthiness of any UI Automation client and therefore the UI Automation provider should not expose protected content or sensitive textual information (such as password fields).
- One of the most significant changes in security for Windows Vista is broadly referred to as "Secure Input" which encompasses technologies such as Least-privileged (or Limited) User Accounts (LUA) and UI Privilege Level Isolation (UIPI).
    - UIPI prevents one program from controlling and/or monitoring another more "privileged" program, preventing cross-process window message attacks that spoof user input.
    - LUA sets limits on the privileges of applications being run by users in the Administrators group. Applications won't necessarily have administrator privileges, but will instead run with the least privileges necessary. As a consequence, there may be some restrictions enforced in LUA scenarios. Most notably string truncation (including TextPattern strings), where it may be necessary to limit the size of strings being retrieved from administrator-level applications so they aren't forced to allocate memory to the point of disabling the application.

Performance

Because TextPattern relies on cross-process calls for most of its functionality, it does not provide a caching mechanism to improve performance when processing content. This is unlike other control patterns in Microsoft UI Automation that can be accessed using the GetCachedPattern or TryGetCachedPattern methods.

One tactic for improving performance is by making sure UI Automation clients attempt to retrieve moderately-sized blocks of text using GetText. For example, GetText(1) calls will incur cross-process hits for each character whereas one GetText(-1) call will incur one cross-process hit, but can have high latency depending on the size of the text provider.

TextPattern Terminology

Attribute
A formatting characteristic of a text range (for example, IsItalicAttribute or FontNameAttribute).

Degenerate Range
A degenerate range is an empty or zero-character text range. For the purposes of the TextPattern control pattern, the text insertion point (or system caret) is considered a degenerate range. If no text is selected, GetSelection would return a degenerate range at the text insertion point and RangeFromPoint would return a degenerate range as its starting endpoint. RangeFromChild and GetVisibleRanges may return degenerate ranges when the text provider cannot find any text ranges that match the given condition. This degenerate range can be used as a starting endpoint within the text provider. FindText and FindAttribute return a null reference (`Nothing` in Microsoft Visual Basic .NET) to avoid confusion with a discovered range versus a degenerate range.

Embedded Object
There are two types of embedded objects in the UI Automation text model. They consist of text-based content elements such as hyperlinks or tables, and control elements such as images and buttons. For more detailed information, see Access Embedded Objects Using UI Automation.

Endpoint
The absolute Start or End point of a text range within a text container.

The absolute starting or ending point
The following illustrates a set of start and end points.

TextRange
A representation of a span of text, with start and end points, in a text container including all associated attributes and functionality.

TextUnit
A pre-defined unit of text (character, word, line, or paragraph) used for navigating through logical segments of a text range.

See also

- UI Automation Control Patterns for Clients
- UI Automation Control Patterns Overview
- UI Automation Tree Overview
- Use Caching in UI Automation
- Support Control Patterns in a UI Automation Provider
- Control Pattern Mapping for UI Automation Clients
- Text Services Framework

TextPattern and Embedded Objects Overview

3/12/2020 • 5 minutes to read • Edit Online

NOTE

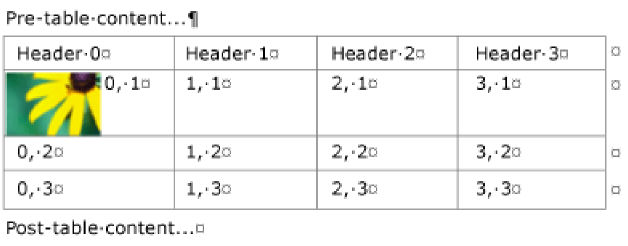
This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This overview describes how Microsoft UI Automation exposes embedded objects, or child elements, within a text document or container.

In UI Automation an embedded object is any element that has non-textual boundaries; for example, an image, hyperlink, table, or document type such as an Microsoft Excel spreadsheet or Microsoft Windows Media file. This differs from the standard definition, where an element is created in one application and embedded, or linked, within another. Whether the object can be edited within its original application is irrelevant in the context of UI Automation.

Embedded Objects and the UI Automation Tree

Embedded objects are treated as individual elements within the control view of the UI Automation tree. They are exposed as children of the text container so that they can be accessed through the same model as other controls in UI Automation.

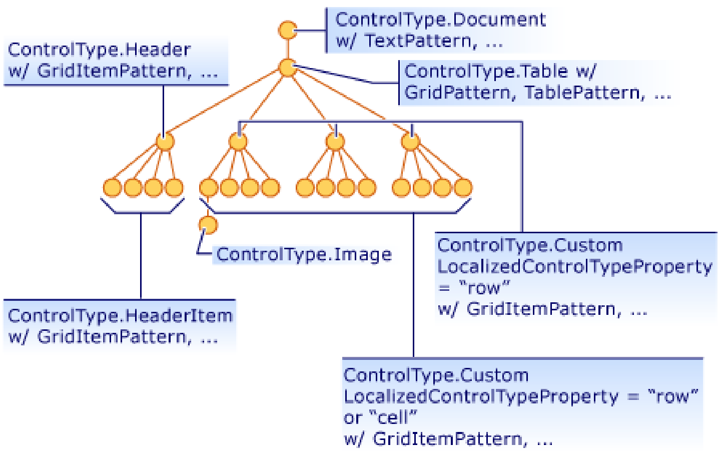


Pre-table·content…¶

Header·0¤	Header·1¤	Header·2¤	Header·3¤	¤
0,·1¤	1,·1¤	2,·1¤	3,·1¤	¤
0,·2¤	1,·2¤	2,·2¤	3,·2¤	¤
0,·3¤	1,·3¤	2,·3¤	3,·3¤	¤

Post-table·content…¤

Example of a Text Container with Table, Image, and Hyperlink Embedded Objects

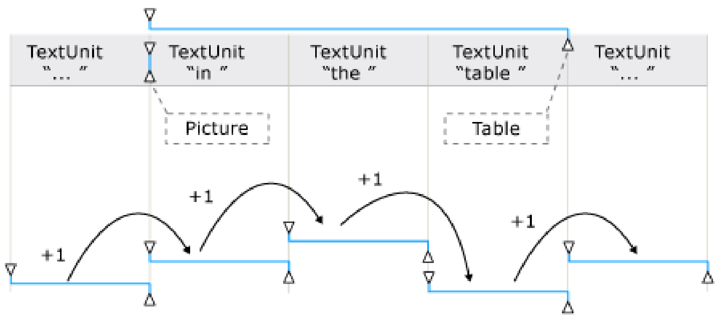


Example of the Content View for a Portion of the Preceding Text Container

Expose Embedded Objects Using TextPattern and TextPatternRange

Used in conjunction, the TextPattern control pattern class and the TextPatternRange class expose methods and properties that facilitate navigation and querying of embedded objects.

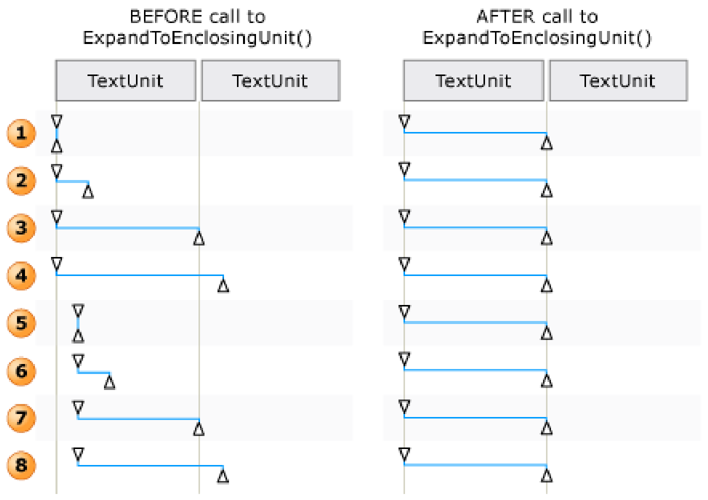
The textual content (or inner text) of a text container and an embedded object, such as a hyperlink or table cell, is exposed as a single, continuous text stream in both the control view and the content view of the UI Automation tree; object boundaries are ignored. If a UI Automation client is retrieving the text for the purpose of reciting, interpreting, or analyzing in some manner, the text range should be checked for special cases, such as a table with textual content or other embedded objects. This can be accomplished by calling GetChildren to obtain an AutomationElement for each embedded object and then calling RangeFromChild to obtain a text range for each element. This is done recursively until all textual content has been retrieved.



Example of a text stream with embedded objects and their range spans

When it is necessary to traverse the content of a text range, a series of steps are involved behind the scenes in order for the Move method to execute successfully.

1. The text range is normalized; that is, the text range is collapsed to a degenerate range at the Start endpoint, which makes the End endpoint superfluous. This step is necessary to remove ambiguity in situations where a text range spans TextUnit boundaries: for example, `{The URL https://www.microsoft.com is embedded in text` where "{" and "}" are the text range endpoints.
2. The resulting range is moved backward in the DocumentRange to the beginning of the requested TextUnit boundary.
3. The range is moved forward or backward in the DocumentRange by the requested number of TextUnit boundaries.
4. The range is then expanded from a degenerate range state by moving the End endpoint by one requested TextUnit boundary.


BEFORE call to ExpandToEnclosingUnit()
AFTER call to ExpandToEnclosingUnit()
TextUnit
TextUnit
TextUnit
TextUnit
1
2
3
4
5
6
7
8

Examples of how a text range is adjusted for Move() and ExpandToEnclosingUnit()

Common Scenarios

The following sections present examples of the most common scenarios that involve embedded objects.

Legend for the examples shown:

{ = Start

} = End

Hyperlink

Example 1 - A text range that contains an embedded text hyperlink

`{The URL https://www.microsoft.com is embedded in text}.`

METHOD CALLED	RESULT
GetText	Returns the string `The URL https://www.microsoft.com is embedded in text` .
GetEnclosingElement	Returns the innermost AutomationElement that encloses the text range; in this case, the AutomationElement that represents the text provider itself.
GetChildren	Returns an AutomationElement representing the hyperlink control.
RangeFromChild where AutomationElement is the object returned by the previous `GetChildren` method.	Returns the range that represents "https://www.microsoft.com".

Example 2 - A text range that partially spans an embedded text hyperlink

The URL `https://{[www]}` is embedded in text.

METHOD CALLED	RESULT
GetText	Returns the string "www".
GetEnclosingElement	Returns the innermost AutomationElement that encloses the text range; in this case, the hyperlink control.
GetChildren	Returns `null` since the text range doesn't span the entire URL string.

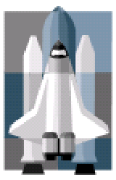
Example 3 - A text range that partially spans the content of a text container. The text container has an embedded text hyperlink that is not part of the text range.

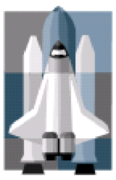
`{The URL} [https://www.microsoft.com](https://www.microsoft.com) is embedded in text.`

METHOD CALLED	RESULT
GetText	Returns the string "The URL".
GetEnclosingElement	Returns the innermost AutomationElement that encloses the text range; in this case, the AutomationElement that represents the text provider itself.
Move with parameters of (TextUnit.Word, 1).	Moves the text range span to "http" since the text of the hyperlink is comprised of individual words. In this case, the hyperlink is not treated as a single object. The URL {[http]} is embedded in text.

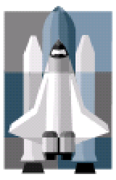
Image

Example 1 - A text range that contains an embedded image

{The image  is embedded in text}.

METHOD CALLED	RESULT
GetText	Returns the string "The is embedded in text". Any ALT text associated with the image cannot be expected to be included in the text stream.
GetEnclosingElement	Returns the innermost AutomationElement that encloses the text range; in this case, the AutomationElement that represents the text provider itself.
GetChildren	Returns an AutomationElement representing the image control.
RangeFromChild where AutomationElement is the object returned by the previous GetChildren method.	Returns the degenerate range that represents "  ".

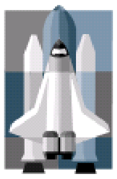
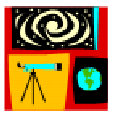
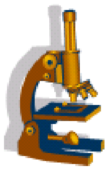
Example 2 - A text range that partially spans the content of a text container. The text container has an embedded image that is not part of the text range.

{The image}  is embedded in text.

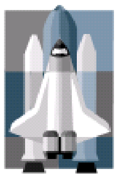
METHOD CALLED	RESULT
GetText	Returns the string "The image".
GetEnclosingElement	Returns the innermost AutomationElement that encloses the text range; in this case, the AutomationElement that represents the text provider itself.
Move with parameters of (TextUnit.Word, 1).	Moves the text range span to "is ". Because only text-based embedded objects are considered part of the text stream, the image in this example does not affect Move or its return value (1 in this case).

Table

Table used for examples

CELL WITH IMAGE	CELL WITH TEXT
	X
	Y
 Image for Z	Z

Example 1 - Get the text container from the content of a cell.

METHOD CALLED	RESULT
GetItem with parameters (0,0)	Returns the AutomationElement representing the content of the table cell; in this case, the element is a text control.
RangeFromChild where AutomationElement is the object returned by the previous `GetItem` method.	Returns the range that spans the image  .
GetEnclosingElement for the object returned by the previous `RangeFromChild` method.	Returns the AutomationElement representing the table cell; in this case, the element is a text control that supports TableItemPattern.
GetEnclosingElement for the object returned by the previous `GetEnclosingElement` method.	Returns the AutomationElement representing the table.
GetEnclosingElement for the object returned by the previous `GetEnclosingElement` method.	Returns the AutomationElement that represents the text provider itself.

Example 2 - Get the text content of a cell.

METHOD CALLED	RESULT
GetItem with parameters of (1,1).	Returns the AutomationElement representing the content of the table cell; in this case, the element is a text control.
RangeFromChild where AutomationElement is the object returned by the previous `GetItem` method.	Returns "Y".

See also

- TextPattern
- TextPatternRange
- ITextProvider
- ITextRangeProvider
- Access Embedded Objects Using UI Automation
- Expose the Content of a Table Using UI Automation
- Traverse Text Using UI Automation
- TextPattern Search and Selection Sample

UI Automation Text Pattern-How-to Topics

12/4/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section includes topics that explain how to use UI Automation text pattern.

In This Section

Add Content to a Text Box Using UI Automation

11/23/2019 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic contains example code that demonstrates how to use Microsoft UI Automation to insert text into a single-line text box. An alternate method is provided for multi-line and rich text controls where UI Automation is not applicable. For comparison purposes, the example also demonstrates how to use Win32 methods to accomplish the same results.

Example

The following example steps through a sequence of text controls in a target application. Each text control is tested to see if a ValuePattern object can be obtained from it using the TryGetCurrentPattern method. If the text control does support ValuePattern, the SetValue method is used to insert a user-defined string into the text control. Otherwise, the SendKeys.SendWait method is used.

```
///--------------------------------------------------------------------
/// <summary>
/// Sets the values of the text controls using managed methods.
/// </summary>
/// <param name="s">The string to be inserted.</param>
///--------------------------------------------------------------------
private void SetValueWithUIAutomation(string s)
{
    foreach (AutomationElement control in textControls)
    {
        InsertTextUsingUIAutomation(control, s);
    }
}

///--------------------------------------------------------------------
/// <summary>
/// Inserts a string into each text control of interest.
/// </summary>
/// <param name="element">A text control.</param>
/// <param name="value">The string to be inserted.</param>
///--------------------------------------------------------------------
private void InsertTextUsingUIAutomation(AutomationElement element,
                                    string value)
{
    try
    {
        // Validate arguments / initial setup
        if (value == null)
            throw new ArgumentNullException(
                "String parameter must not be null.");

        if (element == null)
            throw new ArgumentNullException(
                "AutomationElement parameter must not be null");

        // A series of basic checks prior to attempting an insertion.
        //
        // Check #1: Is control enabled?
        // An alternative to testing for static or read-only controls
        // is to filter using
        // PropertyCondition(AutomationElement.IsEnabledProperty, true)
        // and exclude all read-only text controls from the collection.
        if (!element.Current.IsEnabled)
        {
            throw new InvalidOperationException(
                "The control with an AutomationID of "
                + element.Current.AutomationId.ToString()
                + " is not enabled.\n\n");
        }

        // Check #2: Are there styles that prohibit us
        //           from sending text to this control?
        if (!element.Current.IsKeyboardFocusable)
        {
            throw new InvalidOperationException(
                "The control with an AutomationID of "
                + element.Current.AutomationId.ToString()
                + "is read-only.\n\n");
        }

        // Once you have an instance of an AutomationElement,
        // check if it supports the ValuePattern pattern.
        object valuePattern = null;

        // Control does not support the ValuePattern pattern
        // so use keyboard input to insert content.
        //
        // NOTE: Elements that support TextPattern
        //       do not support ValuePattern and TextPattern
        //       does not support setting the text of
        //       multi-line edit or document controls.
        //       For this reason, text input must be simulated
        //       using one of the following methods.
        //
        if (!element.TryGetCurrentPattern(
            ValuePattern.Pattern, out valuePattern))
        {
            feedbackText.Append("The control with an AutomationID of ")
                .Append(element.Current.AutomationId.ToString())
                .Append(" does not support ValuePattern.")
                .AppendLine(" Using keyboard input.\n");

            // Set focus for input functionality and begin.
            element.SetFocus();

            // Pause before sending keyboard input.
            Thread.Sleep(100);

            // Delete existing content in the control and insert new content.
            SendKeys.SendWait("^{HOME}");   // Move to start of control
            SendKeys.SendWait("^+{END}");   // Select everything
            SendKeys.SendWait("{DEL}");     // Delete selection
            SendKeys.SendWait(value);
        }
        // Control supports the ValuePattern pattern so we can
        // use the SetValue method to insert content.
        else
        {
            feedbackText.Append("The control with an AutomationID of ")
                .Append(element.Current.AutomationId.ToString())
                .Append((" supports ValuePattern."))
                .AppendLine(" Using ValuePattern.SetValue().\n");

            // Set focus for input functionality and begin.
            element.SetFocus();
            element.SetFocus();

            ((ValuePattern)valuePattern).SetValue(value);
        }
    }
    catch (ArgumentNullException exc)
    {
        feedbackText.Append(exc.Message);
    }
    catch (InvalidOperationException exc)
    {
        feedbackText.Append(exc.Message);
    }
    finally
    {
        Feedback(feedbackText.ToString());
    }
}
```

```
'' --------------------------------------------------------------------
''  <summary>
''  Sets the values of the text controls using managed methods.
''  </summary>
''  <param name="s">The string to be inserted.</param>
'' --------------------------------------------------------------------
Private Sub SetValueWithUIAutomation(ByVal s As String)
    Dim control As AutomationElement
    For Each control In textControls
        InsertTextWithUIAutomation(control, s)
    Next control

End Sub


'' --------------------------------------------------------------------
''  <summary>
''  Inserts a string into each text control of interest.
''  </summary>
''  <param name="element">A text control.</param>
''  <param name="value">The string to be inserted.</param>
'' --------------------------------------------------------------------
Private Sub InsertTextWithUIAutomation( _
ByVal element As AutomationElement, ByVal value As String)
    Try
        ' Validate arguments / initial setup
        If value Is Nothing Then
            Throw New ArgumentNullException( _
            "String parameter must not be null.")
        End If

        If element Is Nothing Then
            Throw New ArgumentNullException( _
            "AutomationElement parameter must not be null")
        End If

        ' A series of basic checks prior to attempting an insertion.
        '
        ' Check #1: Is control enabled?
        ' An alternative to testing for static or read-only controls
        ' is to filter using
        ' PropertyCondition(AutomationElement.IsEnabledProperty, true)
        ' and exclude all read-only text controls from the collection.
        If Not element.Current.IsEnabled Then
            Throw New InvalidOperationException( _
            "The control with an AutomationID of " + _
            element.Current.AutomationId.ToString() + _
            " is not enabled." + vbLf + vbLf)
        End If
        ' Check #2: Are there styles that prohibit us
        '           from sending text to this control?
        If Not element.Current.IsKeyboardFocusable Then
            Throw New InvalidOperationException( _
            "The control with an AutomationID of " + _
            element.Current.AutomationId.ToString() + _
            "is read-only." + vbLf + vbLf)
        End If


        ' Once you have an instance of an AutomationElement,
        ' check if it supports the ValuePattern pattern.
        Dim targetValuePattern As Object = Nothing

        ' Control does not support the ValuePattern pattern
        ' so use keyboard input to insert content.
        '
        ' NOTE: Elements that support TextPattern
        '       do not support ValuePattern and TextPattern
        '       does not support setting the text of
        '       multi-line edit or document controls.
        '       For this reason, text input must be simulated
        '       using one of the following methods.
        '
        If Not element.TryGetCurrentPattern(ValuePattern.Pattern, targetValuePattern) Then
            feedbackText.Append("The control with an AutomationID of ") _
            .Append(element.Current.AutomationId.ToString()) _
            .Append(" does not support ValuePattern."). _
            AppendLine(" Using keyboard input.").AppendLine()

            ' Set focus for input functionality and begin.
            element.SetFocus()

            ' Pause before sending keyboard input.
            Thread.Sleep(100)

            ' Delete existing content in the control and insert new content.
            SendKeys.SendWait("^{HOME}") ' Move to start of control
            SendKeys.SendWait("^+{END}") ' Select everything
            SendKeys.SendWait("{DEL}") ' Delete selection
            SendKeys.SendWait(value)
        Else
            ' Control supports the ValuePattern pattern so we can
            ' use the SetValue method to insert content.
            feedbackText.Append("The control with an AutomationID of ") _
            .Append(element.Current.AutomationId.ToString()) _
            .Append(" supports ValuePattern.") _
            .AppendLine(" Using ValuePattern.SetValue().").AppendLine()

            ' Set focus for input functionality and begin.
            element.SetFocus()
            Dim valueControlPattern As ValuePattern = _
            DirectCast(targetValuePattern, ValuePattern)
            valueControlPattern.SetValue(value)
        End If
    Catch exc As ArgumentNullException
        feedbackText.Append(exc.Message)
    Catch exc As InvalidOperationException
        feedbackText.Append(exc.Message)
    Finally
        Feedback(feedbackText.ToString())
    End Try

End Sub
```

See also

- TextPattern Insert Text Sample

Find and Highlight Text Using UI Automation

11/23/2019 • 12 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic demonstrates how to sequentially search for and highlight each occurrence of a string within the content of a text control using Microsoft UI Automation.

Example

The following example obtains a TextPattern object from a text control. A TextPatternRange object, representing the textual content of the entire document, is then created using the DocumentRange property of this TextPattern. Two additional TextPatternRange objects are then created for the sequential search and highlight functionality.

```
///--------------------------------------------------------------------
/// <summary>
/// Starts the target application.
/// </summary>
/// <param name="app">
/// The application to start.
/// </param>
/// <returns>The automation element for the app main window.</returns>
/// <remarks>
/// Three WPF documents, a rich text document, and a plain text document
/// are provided in the Content folder of the TextProvider project.
/// </remarks>
///--------------------------------------------------------------------
private AutomationElement StartApp(string app)
{
    // Start application.
    Process p = Process.Start(app);

    // Give the target application some time to start.
    // For Win32 applications, WaitForInputIdle can be used instead.
    // Another alternative is to listen for WindowOpened events.
    // Otherwise, an ArgumentException results when you try to
    // retrieve an automation element from the window handle.
    Thread.Sleep(2000);

    targetResult.Content =
        WPFTarget +
        " started. \n\nPlease load a document into the target " +
        "application and click the 'Find edit control' button above. " +
        "\n\nNOTE: Documents can be found in the 'Content' folder of the FindText project.";
    targetResult.Background = Brushes.LightGreen;

    // Return the automation element for the app main window.
    return (AutomationElement.FromHandle(p.MainWindowHandle));
}
```

```vb
'--------------------------------------------------------------------
' Starts the target application.
' <param name="app">
' The application to start.
' <returns>The automation element for the app main window.</returns>
' Three WPF documents, a rich text document, and a plain text document
' are provided in the Content folder of the TextProvider project.
'--------------------------------------------------------------------
Private Function StartApp(ByVal app As String) As AutomationElement
    ' Start application.
    Dim p As Process = Process.Start(app)

    ' Give the target application some time to start.
    ' For Win32 applications, WaitForInputIdle can be used instead.
    ' Another alternative is to listen for WindowOpened events.
    ' Otherwise, an ArgumentException results when you try to
    ' retrieve an automation element from the window handle.
    Thread.Sleep(2000)

    targetResult.Content = WPFTarget + " started. " + vbLf + vbLf + _
    "Please load a document into the target application and click " + _
    "the 'Find edit control' button above. " + vbLf + vbLf + _
    "NOTE: Documents can be found in the 'Content' folder of the FindText project."
    targetResult.Background = Brushes.LightGreen

    ' Return the automation element for the app main window.
    Return AutomationElement.FromHandle(p.MainWindowHandle)

End Function 'StartApp
```

```csharp
///--------------------------------------------------------------------
/// <summary>
/// Finds the text control in our target.
/// </summary>
/// <param name="src">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
/// <remarks>
/// Initializes the TextPattern object and event handlers.
/// </remarks>
///--------------------------------------------------------------------
private void FindTextProvider_Click(object src, RoutedEventArgs e)
{
    // Set up the conditions for finding the text control.
    PropertyCondition documentControl = new PropertyCondition(
        AutomationElement.ControlTypeProperty,
        ControlType.Document);
    PropertyCondition textPatternAvailable = new PropertyCondition(
        AutomationElement.IsTextPatternAvailableProperty, true);
    AndCondition findControl =
        new AndCondition(documentControl, textPatternAvailable);

    // Get the Automation Element for the first text control found.
    // For the purposes of this sample it is sufficient to find the
    // first text control. In other cases there may be multiple text
    // controls to sort through.
    targetDocument =
        targetWindow.FindFirst(TreeScope.Descendants, findControl);

    // Didn't find a text control.
    if (targetDocument == null)
    {
        targetResult.Content =
            WPFTarget +
            " does not contain a Document control type.";
        targetResult.Background = Brushes.Salmon;
        startWPFTargetButton.IsEnabled = false;
        return;
    }

    // Get required control patterns
    targetTextPattern =
        targetDocument.GetCurrentPattern(
        TextPattern.Pattern) as TextPattern;

    // Didn't find a text control that supports TextPattern.
    if (targetTextPattern == null)
    {
        targetResult.Content =
            WPFTarget +
            " does not contain an element that supports TextPattern.";
        targetResult.Background = Brushes.Salmon;
        startWPFTargetButton.IsEnabled = false;
        return;
    }

    // Text control is available so display the client controls.
    infoGrid.Visibility = Visibility.Visible;

    targetResult.Content =
        "Text provider found.";
    targetResult.Background = Brushes.LightGreen;

    // Initialize the document range for the text of the document.
    documentRange = targetTextPattern.DocumentRange;

    // Initialize the client's search buttons.
    if (targetTextPattern.DocumentRange.GetText(1).Length > 0)
    {
        searchForwardButton.IsEnabled = true;
    }
    // Initialize the client's search TextBox.
    searchString.IsEnabled = true;

    // Check if the text control supports text selection
    if (targetTextPattern.SupportedTextSelection ==
        SupportedTextSelection.None)
    {
        targetResult.Content = "Unable to select text.";
        targetResult.Background = Brushes.Salmon;
        return;
    }

    // Edit control found so remove the find button from the client.
    findEditButton.Visibility = Visibility.Collapsed;

    // Initialize the client with the current target selection, if any.
    NotifySelectionChanged();

    // Search starts at beginning of doc and goes forward
    searchBackward = false;

    // Initialize a text changed listener.
    // An instance of TextPatternRange will become invalid if
    // one of the following occurs:
    // 1) The text in the provider changes via some user activity.
    // 2) ValuePattern.SetValue is used to programatically change
    // the value of the text in the provider.
    // The only way the client application can detect if the text
    // has changed (to ensure that the ranges are still valid),
    // is by setting a listener for the TextChanged event of
    // the TextPattern. If this event is raised, the client needs
    // to update the targetDocumentRange member data to ensure the
    // user is working with the updated text.
    // Clients must always anticipate the possibility that the text
    // can change underneath them.
    Automation.AddAutomationEventHandler(
        TextPattern.TextChangedEvent,
        targetDocument,
        TreeScope.Element,
        TextChanged);

    // Initialize a selection changed listener.
    // The target selection is reflected in the client.
    Automation.AddAutomationEventHandler(
        TextPattern.TextSelectionChangedEvent,
        targetDocument,
        TreeScope.Element,
        OnTextSelectionChange);
}
```

```
'--------------------------------------------------------------------
' Finds the text control in our target.
' <param name="src">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
' Initializes the TextPattern object and event handlers.
'--------------------------------------------------------------------
Private Sub FindTextProvider_Click( _
ByVal src As Object, ByVal e As RoutedEventArgs)
    ' Set up the conditions for finding the text control.
    Dim documentControl As New PropertyCondition( _
    AutomationElement.ControlTypeProperty, ControlType.Document)
    Dim textPatternAvailable As New PropertyCondition( _
    AutomationElement.IsTextPatternAvailableProperty, True)
    Dim findControl As New AndCondition(documentControl, textPatternAvailable)

    ' Get the Automation Element for the first text control found.
    ' For the purposes of this sample it is sufficient to find the
    ' first text control. In other cases there may be multiple text
    ' controls to sort through.
    targetDocument = targetWindow.FindFirst(TreeScope.Descendants, findControl)

    ' Didn't find a text control.
    If targetDocument Is Nothing Then
        targetResult.Content = _
        WPFTarget + " does not contain a Document control type."
        targetResult.Background = Brushes.Salmon
        startWPFTargetButton.IsEnabled = False
        Return
    End If

    ' Get required control patterns
    targetTextPattern = DirectCast( _
    targetDocument.GetCurrentPattern(TextPattern.Pattern), TextPattern)

    ' Didn't find a text control that supports TextPattern.
    If targetTextPattern Is Nothing Then
        targetResult.Content = WPFTarget + _
        " does not contain an element that supports TextPattern."
        targetResult.Background = Brushes.Salmon
        startWPFTargetButton.IsEnabled = False
        Return
    End If
    ' Text control is available so display the client controls.
    infoGrid.Visibility = Visibility.Visible

    targetResult.Content = "Text provider found."
    targetResult.Background = Brushes.LightGreen

    ' Initialize the document range for the text of the document.
    documentRange = targetTextPattern.DocumentRange

    ' Initialize the client's search buttons.
    If targetTextPattern.DocumentRange.GetText(1).Length > 0 Then
        searchForwardButton.IsEnabled = True
    End If
    ' Initialize the client's search TextBox.
    searchString.IsEnabled = True

    ' Check if the text control supports text selection
    If targetTextPattern.SupportedTextSelection = SupportedTextSelection.None Then
        targetResult.Content = "Unable to select text."
        targetResult.Background = Brushes.Salmon
        Return
    End If

    ' Edit control found so remove the find button from the client.
    findEditButton.Visibility = Visibility.Collapsed

    ' Initialize the client with the current target selection, if any.
    NotifySelectionChanged()

    ' Search starts at beginning of doc and goes forward
    searchBackward = False

    ' Initialize a text changed listener.
    ' An instance of TextPatternRange will become invalid if
    ' one of the following occurs:
    ' 1) The text in the provider changes via some user activity.
    ' 2) ValuePattern.SetValue is used to programatically change
    ' the value of the text in the provider.
    ' The only way the client application can detect if the text
    ' has changed (to ensure that the ranges are still valid),
    ' is by setting a listener for the TextChanged event of
    ' the TextPattern. If this event is raised, the client needs
    ' to update the targetDocumentRange member data to ensure the
    ' user is working with the updated text.
    ' Clients must always anticipate the possibility that the text
    ' can change underneath them.
    Dim onTextChanged As AutomationEventHandler = _
    New AutomationEventHandler(AddressOf TextChanged)
    Automation.AddAutomationEventHandler( _
    TextPattern.TextChangedEvent, targetDocument, TreeScope.Element, onTextChanged)
    ' Initialize a selection changed listener.
    ' The target selection is reflected in the client.
    Dim onSelectionChanged As AutomationEventHandler = _
    New AutomationEventHandler(AddressOf OnTextSelectionChange)
    Automation.AddAutomationEventHandler( _
    TextPattern.TextSelectionChangedEvent, targetDocument, _
    TreeScope.Element, onSelectionChanged)

End Sub
```

```
///--------------------------------------------------------------------
/// <summary>
/// Handles changes to the search text in the client.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
/// <remarks>
/// Reset all controls if user changes search text
/// </remarks>
///--------------------------------------------------------------------
void SearchString_Change(object sender, TextChangedEventArgs e)
{
    int startPoints = documentRange.CompareEndpoints(
        TextPatternRangeEndpoint.Start,
        searchRange,
        TextPatternRangeEndpoint.Start);
    int endPoints = documentRange.CompareEndpoints(
        TextPatternRangeEndpoint.End,
        searchRange,
        TextPatternRangeEndpoint.End);

    // If the starting endpoints of the document range and the search
    // range are equivalent then we can search forward only since the
    // search range is at the start of the document.
    if (startPoints == 0)
    {
        searchForwardButton.IsEnabled = true;
        searchBackwardButton.IsEnabled = false;
    }
    // If the ending endpoints of the document range and the search
    // range are identical then we can search backward only since the
    // search range is at the end of the document.
    else if (endPoints == 0)
    {
        searchForwardButton.IsEnabled = false;
        searchBackwardButton.IsEnabled = true;
    }
    // Otherwise we can search both directions.
    else
    {
        searchForwardButton.IsEnabled = true;
        searchBackwardButton.IsEnabled = true;
    }
}

///--------------------------------------------------------------------
/// <summary>
/// Handles the Search button click.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
/// <remarks>Find the text specified in the text box.</remarks>
///--------------------------------------------------------------------
void SearchDirection_Click(object sender, RoutedEventArgs e)
{
    Button searchDirection = (Button)sender;

    // Are we searching backward through the text control?
    searchBackward =
        ((traversalDirection)searchDirection.Tag == traversalDirection.Backward);

    // Check if search text entered
    if (searchString.Text.Trim() == "")
    {
        targetResult.Content = "No search criteria.";
        targetResult.Background = Brushes.Salmon;
        return;
    }

    // Does target range support text selection?
    if (targetTextPattern.SupportedTextSelection ==
        SupportedTextSelection.None)
    {
        targetResult.Content = "Unable to select text.";
        targetResult.Background = Brushes.Salmon;
        return;
    }
    // Does target range support multiple selections?
    if (targetTextPattern.SupportedTextSelection ==
        SupportedTextSelection.Multiple)
    {
        targetResult.Content = "Multiple selections present.";
        targetResult.Background = Brushes.Salmon;
        return;
    }
    // Clone the document range since we modify the endpoints
    // as we search.
    TextPatternRange documentRangeClone = documentRange.Clone();

    // Move the cloned document range endpoints to enable the
    // selection of the next matching text range.
    TextPatternRange[] selectionRange =
        targetTextPattern.GetSelection();
    if (selectionRange[0] != null)
    {
        if (searchBackward)
        {
            documentRangeClone.MoveEndpointByRange(
                TextPatternRangeEndpoint.End,
                selectionRange[0],
                TextPatternRangeEndpoint.Start);
        }
        else
        {
            documentRangeClone.MoveEndpointByRange(
                TextPatternRangeEndpoint.Start,
                selectionRange[0],
                TextPatternRangeEndpoint.End);
        }
    }

    // Find the text specified in the Search textbox.
    // Clone the search range since we need to modify it.
    TextPatternRange searchRangeClone = searchRange.Clone();
    // backward = false? -- search forward, otherwise backward.
    // ignoreCase = false? -- search is case sensitive.
    searchRange =
        documentRangeClone.FindText(
        searchString.Text, searchBackward, false);

    // Search unsuccessful.
    if (searchRange == null)
    {
        // Search string not found at all.
        if (documentRangeClone.CompareEndpoints(
            TextPatternRangeEndpoint.Start,
            searchRangeClone,
            TextPatternRangeEndpoint.Start) == 0)
        {
            targetResult.Content = "Text not found.";
            targetResult.Background = Brushes.Wheat;
            searchBackwardButton.IsEnabled = false;
            searchForwardButton.IsEnabled = false;
        }
        // End of document (either the start or end of the document
        // range depending on search direction) was reached before
        // finding another occurence of the search string.
        else
        {
            targetResult.Content = "End of document reached.";
            targetResult.Background = Brushes.Wheat;
            if (!searchBackward)
            {
                searchRangeClone.MoveEndpointByRange(
                    TextPatternRangeEndpoint.Start,
                    documentRange,
                    TextPatternRangeEndpoint.End);
                searchBackwardButton.IsEnabled = true;
                searchForwardButton.IsEnabled = false;
            }
            else
            {
                searchRangeClone.MoveEndpointByRange(
                    TextPatternRangeEndpoint.End,
                    documentRange,
                    TextPatternRangeEndpoint.Start);
                searchBackwardButton.IsEnabled = false;
                searchForwardButton.IsEnabled = true;
            }
        }
        searchRange = searchRangeClone;
    }
    // The search string was found.
    else
    {
        targetResult.Content = "Text found.";
        targetResult.Background = Brushes.LightGreen;
    }

    searchRange.Select();
    // Scroll the selection into view and align with top of viewport
    searchRange.ScrollIntoView(true);
    // The WPF target doesn't show selected text as highlighted unless
    // the window has focus.
    targetWindow.SetFocus();
}
```

```
'--------------------------------------------------------------------
' Handles changes to the search text in the client.
' <param name="sender">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
' Reset all controls if user changes search text
'--------------------------------------------------------------------
Private Sub SearchString_Change( _
ByVal sender As Object, ByVal e As TextChangedEventArgs)
    Dim startPoints As Integer = _
    documentRange.CompareEndpoints( _
    TextPatternRangeEndpoint.Start, searchRange, _
    TextPatternRangeEndpoint.Start)
    Dim endPoints As Integer = _
    documentRange.CompareEndpoints(TextPatternRangeEndpoint.End, _
    searchRange, TextPatternRangeEndpoint.End)

    ' If the starting endpoints of the document range and the search
    ' range are equivalent then we can search forward only since the
    ' search range is at the start of the document.
    If startPoints = 0 Then
        searchForwardButton.IsEnabled = True
        searchBackwardButton.IsEnabled = False
        ' If the ending endpoints of the document range and the search
        ' range are identical then we can search backward only since the
        ' search range is at the end of the document.
    ElseIf endPoints = 0 Then
        searchForwardButton.IsEnabled = False
        searchBackwardButton.IsEnabled = True
        ' Otherwise we can search both directions.
    Else
        searchForwardButton.IsEnabled = True
        searchBackwardButton.IsEnabled = True
    End If

End Sub

'--------------------------------------------------------------------
' Handles the Search button click.
' <param name="sender">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
' <remarks>Find the text specified in the text box.</remarks>
'--------------------------------------------------------------------
Private Sub SearchDirection_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim searchDirection As Button = CType(sender, Button)
```

```
' Are we searching backward through the text control?
searchBackward = _
(CType(searchDirection.Tag, traversalDirection) = traversalDirection.Backward)

' Check if search text entered
If searchString.Text.Trim() = "" Then
    targetResult.Content = "No search criteria."
    targetResult.Background = Brushes.Salmon
    Return
End If

' Does target range support text selection?
If targetTextPattern.SupportedTextSelection = SupportedTextSelection.None Then
    targetResult.Content = "Unable to select text."
    targetResult.Background = Brushes.Salmon
    Return
End If
' Does target range support multiple selections?
If targetTextPattern.SupportedTextSelection = SupportedTextSelection.Multiple Then
    targetResult.Content = "Multiple selections present."
    targetResult.Background = Brushes.Salmon
    Return
End If
' Clone the document range since we modify the endpoints
' as we search.
Dim documentRangeClone As TextPatternRange = documentRange.Clone()

' Move the cloned document range endpoints to enable the
' selection of the next matching text range.
Dim selectionRange As TextPatternRange() = targetTextPattern.GetSelection()
If Not (selectionRange(0) Is Nothing) Then
    If searchBackward Then
        documentRangeClone.MoveEndpointByRange( _
        TextPatternRangeEndpoint.End, selectionRange(0), _
        TextPatternRangeEndpoint.Start)
    Else
        documentRangeClone.MoveEndpointByRange( _
        TextPatternRangeEndpoint.Start, selectionRange(0), _
        TextPatternRangeEndpoint.End)
    End If
End If

' Find the text specified in the Search textbox.
' Clone the search range since we need to modify it.
Dim searchRangeClone As TextPatternRange = searchRange.Clone()
' backward = false? -- search forward, otherwise backward.
' ignoreCase = false? -- search is case sensitive.
searchRange = documentRangeClone.FindText(searchString.Text, searchBackward, False)

' Search unsuccessful.
If searchRange Is Nothing Then
    ' Search string not found at all.
    If documentRangeClone.CompareEndpoints( _
    TextPatternRangeEndpoint.Start, searchRangeClone, _
    TextPatternRangeEndpoint.Start) = 0 Then
        targetResult.Content = "Text not found."
        targetResult.Background = Brushes.Wheat
        searchBackwardButton.IsEnabled = False
        searchForwardButton.IsEnabled = False
    Else
        ' End of document (either the start or end of the document
        ' range depending on search direction) was reached before
        ' finding another occurence of the search string.
        targetResult.Content = "End of document reached."
        targetResult.Background = Brushes.Wheat
        If Not searchBackward Then
            searchRangeClone.MoveEndpointByRange( _
            TextPatternRangeEndpoint.Start, documentRange,
                TextPatternRangeEndpoint.End)
                searchBackwardButton.IsEnabled = True
                searchForwardButton.IsEnabled = False
            Else
                searchRangeClone.MoveEndpointByRange( _
                TextPatternRangeEndpoint.End, documentRange, _
                TextPatternRangeEndpoint.Start)
                searchBackwardButton.IsEnabled = False
                searchForwardButton.IsEnabled = True
            End If
        End If
        searchRange = searchRangeClone
    Else
        ' The search string was found.
        targetResult.Content = "Text found."
        targetResult.Background = Brushes.LightGreen
    End If

    searchRange.Select()
    ' Scroll the selection into view and align with top of viewport
    searchRange.ScrollIntoView(True)
    ' The WPF target doesn't show selected text as highlighted unless
    ' the window has focus.
    targetWindow.SetFocus()

End Sub
```

See also

- Find and Highlight Text Using UI Automation

Obtain Text Attributes Using UI Automation

11/23/2019 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to use Microsoft UI Automation to obtain text attributes from a text range. A text range can correspond to the current location of the caret (or degenerate selection) within a document, a contiguous selection of text, a collection of disjoint text selections, or the entire textual content of a document.

Example

The following code example demonstrates how to obtain the FontNameAttribute from a text range.

```
/// --------------------------------------------------------------------
/// <summary>
/// Starts the target application and returns the AutomationElement
/// obtained from the targets window handle.
/// </summary>
/// <param name="exe">
/// The target application.
/// </param>
/// <param name="filename">
/// The text file to be opened in the target application
/// </param>
/// <returns>
/// An AutomationElement representing the target application.
/// </returns>
/// --------------------------------------------------------------------
private AutomationElement StartTarget(string exe, string filename)
{
    // Start text editor and load with a text file.
    Process p = Process.Start(exe, filename);

    // targetApp --> the root AutomationElement.
    AutomationElement targetApp =
        AutomationElement.FromHandle(p.MainWindowHandle);

    return targetApp;
}
```

```
''' --------------------------------------------------------------------
''' <summary>
''' Starts the target application and returns the AutomationElement
''' obtained from the targets window handle.
''' </summary>
''' <param name="exe">
''' The target application.
''' </param>
''' <param name="filename">
''' The text file to be opened in the target application
''' </param>
''' <returns>
''' An AutomationElement representing the target application.
''' </returns>
''' --------------------------------------------------------------------
Private Function StartTarget( _
ByVal exe As String, ByVal filename As String) As AutomationElement
    ' Start text editor and load with a text file.
    Dim p As Process = Process.Start(exe, filename)

    ' targetApp --> the root AutomationElement.
    Dim targetApp As AutomationElement
    targetApp = AutomationElement.FromHandle(p.MainWindowHandle)

    Return targetApp
End Function
```

```
/// --------------------------------------------------------------------
/// <summary>
/// Obtain the text control of interest from the target application.
/// </summary>
/// <param name="targetApp">
/// The target application.
/// </param>
/// <returns>
/// An AutomationElement that represents a text provider..
/// </returns>
/// --------------------------------------------------------------------
private AutomationElement GetTextElement(AutomationElement targetApp)
{
    // The control type we're looking for; in this case 'Document'
    PropertyCondition cond1 =
        new PropertyCondition(
        AutomationElement.ControlTypeProperty,
        ControlType.Document);

    // The control pattern of interest; in this case 'TextPattern'.
    PropertyCondition cond2 =
        new PropertyCondition(
        AutomationElement.IsTextPatternAvailableProperty,
        true);

    AndCondition textCondition = new AndCondition(cond1, cond2);

    AutomationElement targetTextElement =
        targetApp.FindFirst(TreeScope.Descendants, textCondition);

    // If targetText is null then a suitable text control was not found.
    return targetTextElement;
}
```

```vb
''' --------------------------------------------------------------------
''' <summary>
''' Obtain the text control of interest from the target application.
''' </summary>
''' <param name="targetApp">
''' The target application.
''' </param>
''' <returns>
''' An AutomationElement. representing a text control.
''' </returns>
''' --------------------------------------------------------------------
Private Function GetTextElement(ByVal targetApp As AutomationElement) As AutomationElement
    ' The control type we're looking for; in this case 'Document'
    Dim cond1 As PropertyCondition = _
        New PropertyCondition( _
        AutomationElement.ControlTypeProperty, _
        ControlType.Document)

    ' The control pattern of interest; in this case 'TextPattern'.
    Dim cond2 As PropertyCondition = _
        New PropertyCondition( _
        AutomationElement.IsTextPatternAvailableProperty, _
        True)

    Dim textCondition As AndCondition = New AndCondition(cond1, cond2)

    Dim targetTextElement As AutomationElement = _
        targetApp.FindFirst(TreeScope.Descendants, textCondition)

    ' If targetText is null then a suitable text control was not found.
    Return targetTextElement
End Function
```

```
/// --------------------------------------------------------------------
/// <summary>
/// Outputs the FontNameAttribute value for a range of text.
/// </summary>
/// <param name="targetTextElement">
/// The AutomationElment that represents a text control.
/// </param>
/// --------------------------------------------------------------------
private void GetFontNameAttribute(AutomationElement targetTextElement)
{
    TextPattern textPattern =
        targetTextElement.GetCurrentPattern(TextPattern.Pattern) as TextPattern;

    if (textPattern == null)
    {
        // Target control doesn't support TextPattern.
        return;
    }

    // If the target control doesn't support selection then return.
    // Otherwise, get the text attribute for the selected text.
    // If there are currently no selections then the text attribute
    // will be obtained from the insertion point.
    TextPatternRange[] textRanges;
    if (textPattern.SupportedTextSelection == SupportedTextSelection.None)
    {
        return;
    }
    else
    {
        textRanges = textPattern.GetSelection();
    }

    foreach (TextPatternRange textRange in textRanges)
    {
        Object textAttribute =
            textRange.GetAttributeValue(
            TextPattern.FontNameAttribute);

        if (textAttribute == TextPattern.MixedAttributeValue)
        {
            // Returns MixedAttributeValue if the value of the
            // specified attribute varies over the text range.
            Console.WriteLine("Mixed fonts.");
        }
        else if (textAttribute == AutomationElement.NotSupported)
        {
            // Returns NotSupported if the specified attribute is
            // not supported by the provider or the control.
            Console.WriteLine(
                "FontNameAttribute not supported by provider.");
        }
        else
        {
            Console.WriteLine(textAttribute.ToString());
        }
    }
}
```

```vb
''' --------------------------------------------------------------------
''' <summary>
''' Outputs the FontNameAttribute value for a range of text.
''' </summary>
''' <param name="targetTextElement">
''' The AutomationElement. that represents the text provider.
''' </param>
''' --------------------------------------------------------------------
Private Sub GetFontNameAttribute( _
ByVal targetTextElement As AutomationElement)
    Dim targetTextPattern As TextPattern = _
        DirectCast(targetTextElement.GetCurrentPattern( _
        TextPattern.Pattern), TextPattern)

    If (targetTextPattern Is Nothing) Then
        ' Target control doesn't support TextPattern.
        Return
    End If

    ' If the target control doesn't support selection then return.
    ' Otherwise, get the text attribute for the selected text.
    ' If there are currently no selections then the text attribute
    ' will be obtained from the insertion point.
    Dim textRanges() As TextPatternRange
    If (targetTextPattern.SupportedTextSelection = SupportedTextSelection.None) Then
        Return
    Else
        textRanges = targetTextPattern.GetSelection()
    End If

    Dim textRange As TextPatternRange
    For Each textRange In textRanges
        Dim textAttribute As Object = _
            textRange.GetAttributeValue( _
            TextPattern.FontNameAttribute)

        If (textAttribute = TextPattern.MixedAttributeValue) Then
            ' Returns MixedAttributeValue if the value of the
            ' specified attribute varies over the text range.
            Console.WriteLine("Mixed fonts.")
        ElseIf (textAttribute = AutomationElement.NotSupported) Then
            ' Returns NotSupported if the specified attribute is
            ' not supported by the provider or the control.
            Console.WriteLine( _
            "FontNameAttribute not supported by provider.")
        Else
            Console.WriteLine(textAttribute.ToString())
        End If
    Next
End Sub
```

The TextPattern control pattern, in tandem with the TextPatternRange class, supports basic text attributes, properties, and methods. For control-specific functionality that is not supported by TextPattern or TextPatternRange the AutomationElement, class provides methods for a UI Automation client to access the corresponding native object model.

See also

- UI Automation TextPattern Overview
- Add Content to a Text Box Using UI Automation
- Find and Highlight Text Using UI Automation
- UI Automation Control Patterns Overview
- UI Automation Control Patterns for Clients

- Obtain Mixed Text Attribute Details Using UI Automation

Obtain Mixed Text Attribute Details Using UI Automation

11/23/2019 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to use Microsoft UI Automation to obtain text attribute details from a text range that spans multiple attribute values. A text range can correspond to the current location of the caret (or degenerate selection) within a document, a contiguous selection of text, a collection of disjoint text selections, or the entire textual content of a document.

Example

The following code example demonstrates how to obtain the FontNameAttribute from a text range where GetAttributeValue returns a MixedAttributeValue object.

```
///--------------------------------------------------------------------
/// <summary>
/// Display the target selection with attribute details in client.
/// </summary>
/// <param name="selectedText">The current target selection.</param>
///--------------------------------------------------------------------
private void DisplaySelectedTextWithAttributes(string selectedText)
{
    targetSelection.Text = selectedText;
    // We're only interested in the FontNameAttribute for the purposes
    // of this sample.
    targetSelectionAttributes.Text =
        ParseTextRangeByAttribute(
        selectedText, TextPattern.FontNameAttribute);
}

///--------------------------------------------------------------------
/// <summary>
/// Parse the target selection based on the text attribute of interest.
/// </summary>
/// <param name="selectedText">The current target selection.</param>
/// <param name="automationTextAttribute">
/// The text attribute of interest.
/// </param>
/// <returns>
/// A string representing the requested attribute details.
/// </returns>
///--------------------------------------------------------------------
private string ParseTextRangeByAttribute(
    string selectedText,
    AutomationTextAttribute automationTextAttribute)
{
    StringBuilder attributeDetails = new StringBuilder();
    // Initialize the current attribute value.
    string attributeValue = "";
    // Make a copy of the text range.
    TextPatternRange searchRangeClone = searchRange.Clone();
    // Collapse the range to the starting endpoint.
    searchRangeClone.Move(TextUnit.Character, -1);
    // Iterate through the range character by character.
    for (int x = 1; x <= selectedText.Length; x++)
    {
        searchRangeClone.Move(TextUnit.Character, 1);
        // Get the attribute value of the current character.
        string newAttributeValue =
            searchRangeClone.GetAttributeValue(automationTextAttribute).ToString();
        // If the new attribute value is not equal to the old then report
        // the new value along with its location within the range.
        if (newAttributeValue != attributeValue)
        {
            attributeDetails.Append(automationTextAttribute.ProgrammaticName)
                .Append(":\n<")
                .Append(newAttributeValue)
                .Append("> at text range position ")
                .AppendLine(x.ToString());
            attributeValue = newAttributeValue;
        }
    }
    return attributeDetails.ToString();
}
```

```
'--------------------------------------------------------------------
' Display the target selection with attribute details in client.
' <param name="selectedText">The current target selection.</param>
'--------------------------------------------------------------------
Private Sub DisplaySelectedTextWithAttributes(ByVal selectedText As String)
    targetSelection.Text = selectedText
    ' We're only interested in the FontNameAttribute for the purposes
    ' of this sample.
    targetSelectionAttributes.Text = _
        ParseTextRangeByAttribute( _
        selectedText, TextPattern.FontNameAttribute)
End Sub

'--------------------------------------------------------------------
' Parse the target selection based on the text attribute of interest.
' <param name="selectedText">The current target selection.</param>
' <param name="automationTextAttribute">
' The text attribute of interest.
' A string representing the requested attribute details.
'--------------------------------------------------------------------
Function ParseTextRangeByAttribute( _
ByVal selectedText As String, _
ByVal automationTextAttribute As AutomationTextAttribute) As String
    Dim attributeDetails As StringBuilder = New StringBuilder()
    ' Initialize the current attribute value.
    Dim attributeValue As String = ""
    ' Make a copy of the text range.
    Dim searchRangeClone As TextPatternRange = searchRange.Clone()
    ' Collapse the range to the starting endpoint.
    searchRangeClone.Move(TextUnit.Character, -1)
    ' Iterate through the range character by character.
    Dim x As Integer
    For x = 1 To selectedText.Length
        searchRangeClone.Move(TextUnit.Character, 1)
        ' Get the attribute value of the current character.
        Dim newAttributeValue As String = _
            searchRangeClone.GetAttributeValue(automationTextAttribute).ToString()
        ' If the new attribute value is not equal to the old then report
        ' the new value along with its location within the range.
        If (newAttributeValue <> attributeValue) Then
            attributeDetails.Append(automationTextAttribute.ProgrammaticName) _
            .Append(":") _
            .Append(vbLf) _
            .Append("<") _
            .Append(newAttributeValue) _
            .Append("> at text range position ") _
            .AppendLine(x.ToString())
            attributeValue = newAttributeValue
        End If
    Next
    Return attributeDetails.ToString()
End Function
```

The TextPattern control pattern, in tandem with the TextPatternRange class, supports basic text attributes, properties, and methods. For control-specific functionality that is not supported by TextPattern or TextPatternRange, the AutomationElement class provides methods for a UI Automation client to access the corresponding native object model.

See also

- UI Automation TextPattern Overview
- Add Content to a Text Box Using UI Automation
- Find and Highlight Text Using UI Automation

- UI Automation Control Patterns Overview
- UI Automation Control Patterns for Clients
- Obtain Text Attributes Using UI Automation

Traverse Text Using UI Automation

11/23/2019 • 9 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how to use Microsoft UI Automation to traverse the textual content of a document by TextUnit increments.

Example

The following code example demonstrates how to traverse the content of a UI Automation text provider. The Move method moves the Start and End endpoints of a TextPatternRange. This text range is typically a degenerate range representing the text insertion point.

NOTE

Since only text-based embedded objects are considered part of the text stream, embedded objects such as images do not affect `Move` or its return value.

```csharp
///--------------------------------------------------------------------
/// <summary>
/// Starts the target application.
/// </summary>
/// <param name="app">
/// The application to start.
/// </param>
/// <returns>The automation element for the app main window.</returns>
/// <remarks>
/// Three WPF documents, a rich text document, and a plain text document
/// are provided in the Content folder of the TextProvider project.
/// </remarks>
///--------------------------------------------------------------------
private AutomationElement StartApp(string app)
{
    // Start application.
    Process p = Process.Start(app);

    // Give the target application some time to start.
    // For Win32 applications, WaitForInputIdle can be used instead.
    // Another alternative is to listen for WindowOpened events.
    // Otherwise, an ArgumentException results when you try to
    // retrieve an automation element from the window handle.
    Thread.Sleep(2000);

    targetResult.Content =
        WPFTarget +
        " started. \n\nPlease load a document into the target " +
        "application and click the 'Find edit control' button above. " +
        "\n\nNOTE: Documents can be found in the 'Content' folder of the FindText project.";
    targetResult.Background = Brushes.LightGreen;

    // Return the automation element for the app main window.
    return (AutomationElement.FromHandle(p.MainWindowHandle));
}
```

```vb
'--------------------------------------------------------------------
' Starts the target application.
' <param name="app">
' The application to start.
' <returns>The automation element for the app main window.</returns>
' Three WPF documents, a rich text document, and a plain text document
' are provided in the Content folder of the TextProvider project.
'--------------------------------------------------------------------
Private Function StartApp(ByVal app As String) As AutomationElement
    ' Start application.
    Dim p As Process = Process.Start(app)

    ' Give the target application some time to start.
    ' For Win32 applications, WaitForInputIdle can be used instead.
    ' Another alternative is to listen for WindowOpened events.
    ' Otherwise, an ArgumentException results when you try to
    ' retrieve an automation element from the window handle.
    Thread.Sleep(2000)

    targetResult.Content = WPFTarget + " started. " + vbLf + vbLf + _
    "Please load a document into the target application and click " + _
    "the 'Find edit control' button above. " + vbLf + vbLf + _
    "NOTE: Documents can be found in the 'Content' folder of the FindText project."
    targetResult.Background = Brushes.LightGreen

    ' Return the automation element for the app main window.
    Return AutomationElement.FromHandle(p.MainWindowHandle)

End Function 'StartApp
```

```
///--------------------------------------------------------------------
/// <summary>
/// Finds the text control in our target.
/// </summary>
/// <param name="src">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
/// <remarks>
/// Initializes the TextPattern object and event handlers.
/// </remarks>
///--------------------------------------------------------------------
private void FindTextProvider_Click(object src, RoutedEventArgs e)
{
    // Set up the conditions for finding the text control.
    PropertyCondition documentControl = new PropertyCondition(
        AutomationElement.ControlTypeProperty,
        ControlType.Document);
    PropertyCondition textPatternAvailable = new PropertyCondition(
        AutomationElement.IsTextPatternAvailableProperty, true);
    AndCondition findControl =
        new AndCondition(documentControl, textPatternAvailable);

    // Get the Automation Element for the first text control found.
    // For the purposes of this sample it is sufficient to find the
    // first text control. In other cases there may be multiple text
    // controls to sort through.
    targetDocument =
        targetWindow.FindFirst(TreeScope.Descendants, findControl);

    // Didn't find a text control.
    if (targetDocument == null)
    {
        targetResult.Content =
            WPFTarget +
            " does not contain a Document control type.";
        targetResult.Background = Brushes.Salmon;
        startWPFTargetButton.IsEnabled = false;
        return;
    }

    // Get required control patterns
    targetTextPattern =
        targetDocument.GetCurrentPattern(
        TextPattern.Pattern) as TextPattern;

    // Didn't find a text control that supports TextPattern.
    if (targetTextPattern == null)
    {
        targetResult.Content =
            WPFTarget +
            " does not contain an element that supports TextPattern.";
        targetResult.Background = Brushes.Salmon;
        startWPFTargetButton.IsEnabled = false;
        return;
    }

    // Text control is available so display the client controls.
    infoGrid.Visibility = Visibility.Visible;

    targetResult.Content =
        "Text provider found.";
    targetResult.Background = Brushes.LightGreen;

    // Initialize the document range for the text of the document.
    documentRange = targetTextPattern.DocumentRange;

    // Initialize the client's search buttons.
    if (targetTextPattern.DocumentRange.GetText(1).Length > 0)
    {
        searchForwardButton.IsEnabled = true;
    }
    // Initialize the client's search TextBox.
    searchString.IsEnabled = true;

    // Check if the text control supports text selection
    if (targetTextPattern.SupportedTextSelection ==
        SupportedTextSelection.None)
    {
        targetResult.Content = "Unable to select text.";
        targetResult.Background = Brushes.Salmon;
        return;
    }

    // Edit control found so remove the find button from the client.
    findEditButton.Visibility = Visibility.Collapsed;

    // Initialize the client with the current target selection, if any.
    NotifySelectionChanged();

    // Search starts at beginning of doc and goes forward
    searchBackward = false;

    // Initialize a text changed listener.
    // An instance of TextPatternRange will become invalid if
    // one of the following occurs:
    // 1) The text in the provider changes via some user activity.
    // 2) ValuePattern.SetValue is used to programatically change
    // the value of the text in the provider.
    // The only way the client application can detect if the text
    // has changed (to ensure that the ranges are still valid),
    // is by setting a listener for the TextChanged event of
    // the TextPattern. If this event is raised, the client needs
    // to update the targetDocumentRange member data to ensure the
    // user is working with the updated text.
    // Clients must always anticipate the possibility that the text
    // can change underneath them.
    Automation.AddAutomationEventHandler(
        TextPattern.TextChangedEvent,
        targetDocument,
        TreeScope.Element,
        TextChanged);

    // Initialize a selection changed listener.
    // The target selection is reflected in the client.
    Automation.AddAutomationEventHandler(
        TextPattern.TextSelectionChangedEvent,
        targetDocument,
        TreeScope.Element,
        OnTextSelectionChange);
}
```

```
'--------------------------------------------------------------------
' Finds the text control in our target.
' <param name="src">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
' Initializes the TextPattern object and event handlers.
'--------------------------------------------------------------------
Private Sub FindTextProvider_Click( _
ByVal src As Object, ByVal e As RoutedEventArgs)
    ' Set up the conditions for finding the text control.
    Dim documentControl As New PropertyCondition( _
    AutomationElement.ControlTypeProperty, ControlType.Document)
    Dim textPatternAvailable As New PropertyCondition( _
    AutomationElement.IsTextPatternAvailableProperty, True)
    Dim findControl As New AndCondition(documentControl, textPatternAvailable)
```

```
' Get the Automation Element for the first text control found.
' For the purposes of this sample it is sufficient to find the
' first text control. In other cases there may be multiple text
' controls to sort through.
targetDocument = targetWindow.FindFirst(TreeScope.Descendants, findControl)

' Didn't find a text control.
If targetDocument Is Nothing Then
    targetResult.Content = _
    WPFTarget + " does not contain a Document control type."
    targetResult.Background = Brushes.Salmon
    startWPFTargetButton.IsEnabled = False
    Return
End If

' Get required control patterns
targetTextPattern = DirectCast( _
targetDocument.GetCurrentPattern(TextPattern.Pattern), TextPattern)

' Didn't find a text control that supports TextPattern.
If targetTextPattern Is Nothing Then
    targetResult.Content = WPFTarget + _
    " does not contain an element that supports TextPattern."
    targetResult.Background = Brushes.Salmon
    startWPFTargetButton.IsEnabled = False
    Return
End If
' Text control is available so display the client controls.
infoGrid.Visibility = Visibility.Visible

targetResult.Content = "Text provider found."
targetResult.Background = Brushes.LightGreen

' Initialize the document range for the text of the document.
documentRange = targetTextPattern.DocumentRange

' Initialize the client's search buttons.
If targetTextPattern.DocumentRange.GetText(1).Length > 0 Then
    searchForwardButton.IsEnabled = True
End If
' Initialize the client's search TextBox.
searchString.IsEnabled = True

' Check if the text control supports text selection
If targetTextPattern.SupportedTextSelection = SupportedTextSelection.None Then
    targetResult.Content = "Unable to select text."
    targetResult.Background = Brushes.Salmon
    Return
End If

' Edit control found so remove the find button from the client.
findEditButton.Visibility = Visibility.Collapsed

' Initialize the client with the current target selection, if any.
NotifySelectionChanged()

' Search starts at beginning of doc and goes forward
searchBackward = False

' Initialize a text changed listener.
' An instance of TextPatternRange will become invalid if
' one of the following occurs:
' 1) The text in the provider changes via some user activity.
' 2) ValuePattern.SetValue is used to programatically change
' the value of the text in the provider.
' The only way the client application can detect if the text
' has changed (to ensure that the ranges are still valid),
' is by setting a listener for the TextChanged event of
' the TextPattern. If this event is raised, the client needs
```

```
    ' to update the targetDocumentRange member data to ensure the
    ' user is working with the updated text.
    ' Clients must always anticipate the possibility that the text
    ' can change underneath them.
    Dim onTextChanged As AutomationEventHandler = _
    New AutomationEventHandler(AddressOf TextChanged)
    Automation.AddAutomationEventHandler( _
    TextPattern.TextChangedEvent, targetDocument, TreeScope.Element, onTextChanged)
    ' Initialize a selection changed listener.
    ' The target selection is reflected in the client.
    Dim onSelectionChanged As AutomationEventHandler = _
    New AutomationEventHandler(AddressOf OnTextSelectionChange)
    Automation.AddAutomationEventHandler( _
    TextPattern.TextSelectionChangedEvent, targetDocument, _
    TreeScope.Element, onSelectionChanged)

End Sub
```

```
///--------------------------------------------------------------------
/// <summary>
/// Handles the navigation item selected event.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
///--------------------------------------------------------------------
private void NavigationUnit_Change(object sender, SelectionChangedEventArgs e)
{
    ComboBox cb = (ComboBox)sender;
    navigationUnit = (TextUnit)cb.SelectedValue;
}

///--------------------------------------------------------------------
/// <summary>
/// Handles the Navigate button click event.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
///--------------------------------------------------------------------
private void Navigate_Click(object sender, RoutedEventArgs e)
{
    Button moveSelection = (Button)sender;

    // Which direction is the user searching through the text control?
    int navDirection =
        ((traversalDirection)moveSelection.Tag == traversalDirection.Forward) ? 1 : -1;

    // Obtain the ranges to move.
    TextPatternRange[] selectionRanges =
                   targetTextPattern.GetSelection();

    // Iterate throught the ranges for a text control that supports
    // multiple selections and move the selections the specified text
    // unit and direction.
    foreach (TextPatternRange textRange in selectionRanges)
    {
        textRange.Move(navigationUnit, navDirection);
        textRange.Select();
   }
   // The WPF target doesn't show selected text as highlighted unless
   // the window has focus.
   targetDocument.SetFocus();
}
```

```
'--------------------------------------------------------------------
' Handles the navigation item selected event.
' <param name="sender">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
'--------------------------------------------------------------------
Private Sub NavigationUnit_Change( _
ByVal sender As Object, ByVal e As SelectionChangedEventArgs)
    Dim cb As ComboBox = CType(sender, ComboBox)
    navigationUnit = CType(cb.SelectedValue, TextUnit)

End Sub

'--------------------------------------------------------------------
' Handles the Navigate button click event.
' <param name="sender">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
'--------------------------------------------------------------------
Private Sub Navigate_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim moveSelection As Button = CType(sender, Button)
    Dim navDirection As Integer

    ' Which direction is the user searching through the text control?
    If (CType(moveSelection.Tag, traversalDirection) = traversalDirection.Forward) Then
        navDirection = 1
    Else
        navDirection = -1
    End If

    ' Obtain the ranges to move.
    Dim selectionRanges As TextPatternRange() = targetTextPattern.GetSelection()

    ' Iterate throught the ranges for a text control that supports
    ' multiple selections and move the selections the specified text
    ' unit and direction.
    Dim textRange As TextPatternRange
    For Each textRange In selectionRanges
        textRange.Move(navigationUnit, navDirection)
        textRange.Select()
    Next textRange
    ' The WPF target doesn't show selected text as highlighted unless
    ' the window has focus.
    targetDocument.SetFocus()

End Sub
```

Any method using TextUnit will defer to the next largest TextUnit supported if the given TextUnit is not supported by the control.

See also

- UI Automation TextPattern Overview
- Add Content to a Text Box Using UI Automation
- Find and Highlight Text Using UI Automation
- UI Automation Control Patterns Overview
- UI Automation Control Patterns for Clients

Access Embedded Objects Using UI Automation

11/23/2019 • 12 minutes to read • Edit Online

NOTE

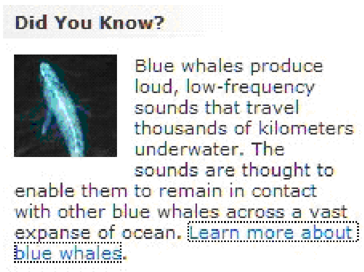
This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic shows how Microsoft UI Automation can be used to expose objects embedded within the content of a text control.

NOTE

Embedded objects can include images, hyperlinks, buttons, tables, or ActiveX controls.

Embedded objects are considered children of the UI Automation text provider. This allows them to be exposed through the same UI Automation tree structure as all other user interface (UI) elements. Functionality, in turn, is exposed through the control patterns typically required by the embedded objects control type (for example, since hyperlinks are text-based they will support TextPattern).



Did You Know?

Blue whales produce loud, low-frequency sounds that travel thousands of kilometers underwater. The sounds are thought to enable them to remain in contact with other blue whales across a vast expanse of ocean. Learn more about blue whales.

A sample document with textual content, ("Did You Know?"...) and two embedded objects (a picture of a whale and a text hyperlink), used as a target for the code examples.

Example

The following code example demonstrates how to retrieve a collection of embedded objects from within a UI Automation text provider. For the sample document provided in the introduction, two objects would be returned (an image element and a text element).

NOTE

The image element should have some intrinsic text associated with it that describes the image, typically in its NameProperty (for example, "A blue whale."). However, when a text range spanning the image object is obtained, neither the image nor this descriptive text is returned in the text stream.

```csharp
///--------------------------------------------------------------------
/// <summary>
/// Starts the target application.
/// </summary>
/// <param name="app">
/// The application to start.
/// </param>
/// <returns>The automation element for the app main window.</returns>
/// <remarks>
/// Three WPF documents, a rich text document, and a plain text document
/// are provided in the Content folder of the TextProvider project.
/// </remarks>
///--------------------------------------------------------------------
private AutomationElement StartApp(string app)
{
    // Start application.
    Process p = Process.Start(app);

    // Give the target application some time to start.
    // For Win32 applications, WaitForInputIdle can be used instead.
    // Another alternative is to listen for WindowOpened events.
    // Otherwise, an ArgumentException results when you try to
    // retrieve an automation element from the window handle.
    Thread.Sleep(2000);

    targetResult.Content =
        WPFTarget +
        " started. \n\nPlease load a document into the target " +
        "application and click the 'Find edit control' button above. " +
        "\n\nNOTE: Documents can be found in the 'Content' folder of the FindText project.";
    targetResult.Background = Brushes.LightGreen;

    // Return the automation element for the app main window.
    return (AutomationElement.FromHandle(p.MainWindowHandle));
}
```

```vb
'--------------------------------------------------------------------
' Starts the target application.
' <param name="app">
' The application to start.
' <returns>The automation element for the app main window.</returns>
' Three WPF documents, a rich text document, and a plain text document
' are provided in the Content folder of the TextProvider project.
'--------------------------------------------------------------------
Private Function StartApp(ByVal app As String) As AutomationElement
    ' Start application.
    Dim p As Process = Process.Start(app)

    ' Give the target application some time to start.
    ' For Win32 applications, WaitForInputIdle can be used instead.
    ' Another alternative is to listen for WindowOpened events.
    ' Otherwise, an ArgumentException results when you try to
    ' retrieve an automation element from the window handle.
    Thread.Sleep(2000)

    targetResult.Content = WPFTarget + " started. " + vbLf + vbLf + _
    "Please load a document into the target application and click " + _
    "the 'Find edit control' button above. " + vbLf + vbLf + _
    "NOTE: Documents can be found in the 'Content' folder of the FindText project."
    targetResult.Background = Brushes.LightGreen

    ' Return the automation element for the app main window.
    Return AutomationElement.FromHandle(p.MainWindowHandle)

End Function 'StartApp
```

```
///--------------------------------------------------------------------
/// <summary>
/// Finds the text control in our target.
/// </summary>
/// <param name="src">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
/// <remarks>
/// Initializes the TextPattern object and event handlers.
/// </remarks>
///--------------------------------------------------------------------
private void FindTextProvider_Click(object src, RoutedEventArgs e)
{
    // Set up the conditions for finding the text control.
    PropertyCondition documentControl = new PropertyCondition(
        AutomationElement.ControlTypeProperty,
        ControlType.Document);
    PropertyCondition textPatternAvailable = new PropertyCondition(
        AutomationElement.IsTextPatternAvailableProperty, true);
    AndCondition findControl =
        new AndCondition(documentControl, textPatternAvailable);

    // Get the Automation Element for the first text control found.
    // For the purposes of this sample it is sufficient to find the
    // first text control. In other cases there may be multiple text
    // controls to sort through.
    targetDocument =
        targetWindow.FindFirst(TreeScope.Descendants, findControl);

    // Didn't find a text control.
    if (targetDocument == null)
    {
        targetResult.Content =
            WPFTarget +
            " does not contain a Document control type.";
        targetResult.Background = Brushes.Salmon;
        startWPFTargetButton.IsEnabled = false;
        return;
    }

    // Get required control patterns
    targetTextPattern =
        targetDocument.GetCurrentPattern(
        TextPattern.Pattern) as TextPattern;

    // Didn't find a text control that supports TextPattern.
    if (targetTextPattern == null)
    {
        targetResult.Content =
            WPFTarget +
            " does not contain an element that supports TextPattern.";
        targetResult.Background = Brushes.Salmon;
        startWPFTargetButton.IsEnabled = false;
        return;
    }

    // Text control is available so display the client controls.
    infoGrid.Visibility = Visibility.Visible;

    targetResult.Content =
        "Text provider found.";
    targetResult.Background = Brushes.LightGreen;

    // Initialize the document range for the text of the document.
    documentRange = targetTextPattern.DocumentRange;

    // Initialize the client's search buttons.
    if (targetTextPattern.DocumentRange.GetText(1).Length > 0)
    {
        searchForwardButton.IsEnabled = true;
    }
    // Initialize the client's search TextBox.
    searchString.IsEnabled = true;

    // Check if the text control supports text selection
    if (targetTextPattern.SupportedTextSelection ==
        SupportedTextSelection.None)
    {
        targetResult.Content = "Unable to select text.";
        targetResult.Background = Brushes.Salmon;
        return;
    }

    // Edit control found so remove the find button from the client.
    findEditButton.Visibility = Visibility.Collapsed;

    // Initialize the client with the current target selection, if any.
    NotifySelectionChanged();

    // Search starts at beginning of doc and goes forward
    searchBackward = false;

    // Initialize a text changed listener.
    // An instance of TextPatternRange will become invalid if
    // one of the following occurs:
    // 1) The text in the provider changes via some user activity.
    // 2) ValuePattern.SetValue is used to programatically change
    // the value of the text in the provider.
    // The only way the client application can detect if the text
    // has changed (to ensure that the ranges are still valid),
    // is by setting a listener for the TextChanged event of
    // the TextPattern. If this event is raised, the client needs
    // to update the targetDocumentRange member data to ensure the
    // user is working with the updated text.
    // Clients must always anticipate the possibility that the text
    // can change underneath them.
    Automation.AddAutomationEventHandler(
        TextPattern.TextChangedEvent,
        targetDocument,
        TreeScope.Element,
        TextChanged);

    // Initialize a selection changed listener.
    // The target selection is reflected in the client.
    Automation.AddAutomationEventHandler(
        TextPattern.TextSelectionChangedEvent,
        targetDocument,
        TreeScope.Element,
        OnTextSelectionChange);
}
```

```
'--------------------------------------------------------------------
' Finds the text control in our target.
' <param name="src">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
' Initializes the TextPattern object and event handlers.
'--------------------------------------------------------------------
Private Sub FindTextProvider_Click( _
ByVal src As Object, ByVal e As RoutedEventArgs)
    ' Set up the conditions for finding the text control.
    Dim documentControl As New PropertyCondition( _
    AutomationElement.ControlTypeProperty, ControlType.Document)
    Dim textPatternAvailable As New PropertyCondition( _
    AutomationElement.IsTextPatternAvailableProperty, True)
    Dim findControl As New AndCondition(documentControl, textPatternAvailable)
    ' Get the Automation Element for the first text control found.
    ' For the purposes of this sample it is sufficient to find the
    ' first text control. In other cases there may be multiple text
    ' controls to sort through.
    targetDocument = targetWindow.FindFirst(TreeScope.Descendants, findControl)

    ' Didn't find a text control.
    If targetDocument Is Nothing Then
        targetResult.Content = _
        WPFTarget + " does not contain a Document control type."
        targetResult.Background = Brushes.Salmon
        startWPFTargetButton.IsEnabled = False
        Return
    End If

    ' Get required control patterns
    targetTextPattern = DirectCast( _
    targetDocument.GetCurrentPattern(TextPattern.Pattern), TextPattern)

    ' Didn't find a text control that supports TextPattern.
    If targetTextPattern Is Nothing Then
        targetResult.Content = WPFTarget + _
        " does not contain an element that supports TextPattern."
        targetResult.Background = Brushes.Salmon
        startWPFTargetButton.IsEnabled = False
        Return
    End If
    ' Text control is available so display the client controls.
    infoGrid.Visibility = Visibility.Visible

    targetResult.Content = "Text provider found."
    targetResult.Background = Brushes.LightGreen

    ' Initialize the document range for the text of the document.
    documentRange = targetTextPattern.DocumentRange

    ' Initialize the client's search buttons.
    If targetTextPattern.DocumentRange.GetText(1).Length > 0 Then
        searchForwardButton.IsEnabled = True
    End If
    ' Initialize the client's search TextBox.
    searchString.IsEnabled = True

    ' Check if the text control supports text selection
    If targetTextPattern.SupportedTextSelection = SupportedTextSelection.None Then
        targetResult.Content = "Unable to select text."
        targetResult.Background = Brushes.Salmon
        Return
    End If

    ' Edit control found so remove the find button from the client.
    findEditButton.Visibility = Visibility.Collapsed

    ' Initialize the client with the current target selection, if any.
    NotifySelectionChanged()

    ' Search starts at beginning of doc and goes forward
    searchBackward = False

    ' Initialize a text changed listener.
    ' An instance of TextPatternRange will become invalid if
    ' one of the following occurs:
    ' 1) The text in the provider changes via some user activity.
    ' 2) ValuePattern.SetValue is used to programatically change
    ' the value of the text in the provider.
    ' The only way the client application can detect if the text
    ' has changed (to ensure that the ranges are still valid),
    ' is by setting a listener for the TextChanged event of
    ' the TextPattern. If this event is raised, the client needs
    ' to update the targetDocumentRange member data to ensure the
    ' user is working with the updated text.
    ' Clients must always anticipate the possibility that the text
    ' can change underneath them.
    Dim onTextChanged As AutomationEventHandler = _
    New AutomationEventHandler(AddressOf TextChanged)
    Automation.AddAutomationEventHandler( _
    TextPattern.TextChangedEvent, targetDocument, TreeScope.Element, onTextChanged)
    ' Initialize a selection changed listener.
    ' The target selection is reflected in the client.
    Dim onSelectionChanged As AutomationEventHandler = _
    New AutomationEventHandler(AddressOf OnTextSelectionChange)
    Automation.AddAutomationEventHandler( _
    TextPattern.TextSelectionChangedEvent, targetDocument, _
    TreeScope.Element, onSelectionChanged)

End Sub
```

```
///--------------------------------------------------------------------
/// <summary>
/// Gets the children of the target selection.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">Event arguments.</param>
///--------------------------------------------------------------------
private void GetChildren_Click(object sender, RoutedEventArgs e)
{
    // Obtain an array of child elements.
    AutomationElement[] textProviderChildren;
    try
    {
        textProviderChildren = searchRange.GetChildren();
    }
    catch (ElementNotAvailableException)
    {
        // TODO: error handling.
        return;
    }

    // Assemble the information about the enclosing element.
    StringBuilder childInformation = new StringBuilder();
    childInformation.Append(textProviderChildren.Length)
        .AppendLine(" child element(s).");

    // Iterate through the collection of child elements and obtain
    // information of interest about each.
    for (int i = 0; i < textProviderChildren.Length; i++)
    {
        childInformation.Append("Child").Append(i).AppendLine(":");
        // Obtain the name of the child control.
        childInformation.Append("\tName:\t")
            .AppendLine(textProviderChildren[i].Current.Name);
        // Obtain the control type.
        childInformation.Append("\tControl Type:\t")
            .AppendLine(textProviderChildren[i].Current.ControlType.ProgrammaticName);

        // Obtain the supported control patterns.
        // NOTE: For the purposes of this sample we use GetSupportedPatterns().
        // However, the use of GetSupportedPatterns() is strongly discouraged
        // as it calls GetCurrentPattern() internally on every known pattern.
        // It is therefore much more efficient to use GetCurrentPattern() for
        // the specific patterns you are interested in.
        AutomationPattern[] childPatterns =
            textProviderChildren[i].GetSupportedPatterns();
        childInformation.AppendLine("\tSupported Control Patterns:");
        if (childPatterns.Length <= 0)
        {
            childInformation.AppendLine("\t\t\tNone");
        }
        else
        {
            foreach (AutomationPattern pattern in childPatterns)
            {
                childInformation.Append("\t\t\t")
                    .AppendLine(pattern.ProgrammaticName);
            }
        }

        // Obtain the child elements, if any, of the child control.
        TextPatternRange childRange =
            documentRange.TextPattern.RangeFromChild(textProviderChildren[i]);
        AutomationElement[] childRangeChildren =
            childRange.GetChildren();
        childInformation.Append("\tChildren: \t").Append(childRangeChildren.Length).AppendLine();
    }
    // Display the information about the child controls.
    targetSelectionDetails.Text = childInformation.ToString();
}
```

```vb
'--------------------------------------------------------------------
' Gets the children of the target selection.
' <param name="sender">The object that raised the event.</param>
' <param name="e">Event arguments.</param>
'--------------------------------------------------------------------
Private Sub GetChildren_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ' Obtain an array of child elements.
    Dim textProviderChildren() As AutomationElement
    Try
        textProviderChildren = searchRange.GetChildren()
    Catch
        ' TODO: error handling.
        Return
    End Try

    ' Assemble the information about the enclosing element.
    Dim childInformation As New StringBuilder()
    childInformation.Append(textProviderChildren.Length).AppendLine(" child element(s).")

    ' Iterate through the collection of child elements and obtain
    ' information of interest about each.
    Dim i As Integer
    For i = 0 To textProviderChildren.Length - 1
        childInformation.Append("Child").Append(i).AppendLine(":")
        ' Obtain the name of the child control.
        childInformation.Append(vbTab + "Name:" + vbTab) _
        .AppendLine(textProviderChildren(i).Current.Name)
        ' Obtain the control type.
        childInformation.Append(vbTab + "Control Type:" + vbTab) _
        .AppendLine(textProviderChildren(i).Current.ControlType.ProgrammaticName)

        ' Obtain the supported control patterns.
        ' NOTE: For the purposes of this sample we use GetSupportedPatterns().
        ' However, the use of GetSupportedPatterns() is strongly discouraged
        ' as it calls GetCurrentPattern() internally on every known pattern.
        ' It is therefore much more efficient to use GetCurrentPattern() for
        ' the specific patterns you are interested in.
        Dim childPatterns As AutomationPattern() = _
        textProviderChildren(i).GetSupportedPatterns()
        childInformation.AppendLine(vbTab + "Supported Control Patterns:")
        If childPatterns.Length <= 0 Then
            childInformation.AppendLine(vbTab + vbTab + vbTab + "None")
        Else
            Dim pattern As AutomationPattern
            For Each pattern In childPatterns
                childInformation.Append(vbTab + vbTab + vbTab) _
                .AppendLine(pattern.ProgrammaticName)
            Next pattern
        End If

        ' Obtain the child elements, if any, of the child control.
        Dim childRange As TextPatternRange = _
        documentRange.TextPattern.RangeFromChild(textProviderChildren(i))
        Dim childRangeChildren As AutomationElement() = _
        childRange.GetChildren()
        childInformation.Append(vbTab + "Children: " + vbTab) _
        .Append(childRangeChildren.Length).AppendLine()
    Next i
    ' Display the information about the child controls.
    targetSelectionDetails.Text = childInformation.ToString()

End Sub
```

Example

The following code example demonstrates how to obtain a text range from an embedded object within a UI Automation text provider. The text range retrieved is an empty range where the starting endpoint follows "... ocean.(space)" and the ending endpoint precedes the closing "." representing the embedded hyperlink (as shown by the image provided in the introduction). Even though this is an empty range, it is not considered a degenerate range because it has a non-zero span.

NOTE

TextPattern can retrieve a text-based embedded object such as a hyperlink; however, a secondary TextPattern will have to be obtained from the embedded object to expose its full functionality.

```
/// -------------------------------------------------------------------
/// <summary>
/// Obtains a text range spanning an embedded child
/// of a document control and displays the content of the range.
/// </summary>
/// <param name="targetTextElement">
/// The AutomationElment that represents a text control.
/// </param>
/// -------------------------------------------------------------------
private void GetRangeFromChild(AutomationElement targetTextElement)
{
    TextPattern textPattern =
        targetTextElement.GetCurrentPattern(TextPattern.Pattern)
        as TextPattern;

    if (textPattern == null)
    {
        // Target control doesn't support TextPattern.
        return;
    }

    // Obtain a text range spanning the entire document.
    TextPatternRange textRange = textPattern.DocumentRange;

    // Retrieve the embedded objects within the range.
    AutomationElement[] embeddedObjects = textRange.GetChildren();

    // Retrieve and display text value of embedded object.
    foreach (AutomationElement embeddedObject in embeddedObjects)
    {
        if ((bool)embeddedObject.GetCurrentPropertyValue(
            AutomationElement.IsTextPatternAvailableProperty))
        {
           // For full functionality a secondary TextPattern should
           // be obtained from the embedded object.
           // embeddedObject must be a child of the text provider.
            TextPatternRange embeddedObjectRange =
                textPattern.RangeFromChild(embeddedObject);
            // GetText(-1) retrieves all text in the range.
            // Typically a more limited amount of text would be
            // retrieved for performance and security reasons.
            Console.WriteLine(embeddedObjectRange.GetText(-1));
        }
    }
}
```

```
''' --------------------------------------------------------------------
''' <summary>
''' Obtains a text range spanning an embedded child
''' of a document control and displays the content of the range.
''' </summary>
''' <param name="targetTextElement">
''' The AutomationElement. representing a text control.
''' </param>
''' --------------------------------------------------------------------
Private Sub GetRangeFromChild( _
ByVal targetTextElement As AutomationElement)
    Dim textPattern As TextPattern = _
    DirectCast( _
    targetTextElement.GetCurrentPattern(textPattern.Pattern), _
    TextPattern)

    If (textPattern Is Nothing) Then
        ' Target control doesn't support TextPattern.
        Return
    End If

    ' Obtain a text range spanning the entire document.
    Dim textRange As TextPatternRange = textPattern.DocumentRange

    ' Retrieve the embedded objects within the range.
    Dim embeddedObjects() As AutomationElement = textRange.GetChildren()

    Dim embeddedObject As AutomationElement
    For Each embeddedObject In embeddedObjects
        If (embeddedObject.GetCurrentPropertyValue( _
            AutomationElement.IsTextPatternAvailableProperty) = True) Then
            ' For full functionality a secondary TextPattern should
            ' be obtained from the embedded object.
            ' embeddedObject must be a child of the text provider.
            Dim embeddedObjectRange As TextPatternRange = _
            textPattern.RangeFromChild(embeddedObject)
            ' GetText(-1) retrieves all text in the range.
            ' Typically a more limited amount of text would be
            ' retrieved for performance and security reasons.
            Console.WriteLine(embeddedObjectRange.GetText(-1))
        End If
    Next
End Sub
```

See also

- UI Automation TextPattern Overview
- UI Automation Control Patterns Overview
- UI Automation Control Patterns for Clients
- Add Content to a Text Box Using UI Automation
- Find and Highlight Text Using UI Automation

UI Automation Control Types

11/23/2019 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This section contains information about support for control types in Microsoft UI Automation.

In This Section

- UI Automation Control Types Overview
- UI Automation Support for the Button Control Type
- UI Automation Support for the Calendar Control Type
- UI Automation Support for the CheckBox Control Type
- UI Automation Support for the ComboBox Control Type
- UI Automation Support for the DataGrid Control Type
- UI Automation Support for the DataItem Control Type
- UI Automation Support for the Document Control Type
- UI Automation Support for the Edit Control Type
- UI Automation Support for the Group Control Type
- UI Automation Support for the Header Control Type
- UI Automation Support for the HeaderItem Control Type
- UI Automation Support for the Hyperlink Control Type
- UI Automation Support for the Image Control Type
- UI Automation Support for the List Control Type
- UI Automation Support for the ListItem Control Type
- UI Automation Support for the Menu Control Type
- UI Automation Support for the MenuBar Control Type
- UI Automation Support for the MenuItem Control Type
- UI Automation Support for the Pane Control Type
- UI Automation Support for the ProgressBar Control Type
- UI Automation Support for the RadioButton Control Type
- UI Automation Support for the ScrollBar Control Type
- UI Automation Support for the Separator Control Type
- UI Automation Support for the Slider Control Type
- UI Automation Support for the Spinner Control Type
- UI Automation Support for the SplitButton Control Type
- UI Automation Support for the StatusBar Control Type
- UI Automation Support for the Tab Control Type
- UI Automation Support for the TabItem Control Type
- UI Automation Support for the Table Control Type

- UI Automation Support for the Text Control Type
- UI Automation Support for the Thumb Control Type
- UI Automation Support for the TitleBar Control Type
- UI Automation Support for the ToolBar Control Type
- UI Automation Support for the ToolTip Control Type
- UI Automation Support for the Tree Control Type
- UI Automation Support for the TreeItem Control Type
- UI Automation Support for the Window Control Type

Reference

ControlType

See also

- UI Automation Control Patterns

UI Automation Control Types Overview

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Microsoft UI Automation control types are well-known identifiers that can be used to indicate what kind of control a particular element represents, such as a combo box or a button.

Having a well-known identifier makes it easier for assistive technology devices to determine what types of controls are available in the user interface (UI) and how to interact with the controls.

UI Automation Control Type Requisites

Microsoft UI Automation control types provide a set of conditions that providers must meet. When these conditions are met, the control can use the specific control type name. Each control type has conditions for the following:

- UI Automation control patterns—which control patterns must be supported, which control patterns are optional, and which control patterns must not be supported by the control.
- UI Automation property values—which property values are supported.
- UI Automation tree structure—the required UI Automation tree structure for the control.

When a control meets the conditions for a particular control type, the ControlType property value will indicate that control type.

Current UI Automation Control Types

The following list contains the current set of Microsoft UI Automation control types:

- UI Automation Support for the Button Control Type
- UI Automation Support for the Calendar Control Type
- UI Automation Support for the CheckBox Control Type
- UI Automation Support for the ComboBox Control Type
- UI Automation Support for the DataGrid Control Type
- UI Automation Support for the DataItem Control Type
- UI Automation Support for the Document Control Type
- UI Automation Support for the Edit Control Type
- UI Automation Support for the Group Control Type
- UI Automation Support for the Header Control Type

- UI Automation Support for the HeaderItem Control Type
- UI Automation Support for the Hyperlink Control Type
- UI Automation Support for the Image Control Type
- UI Automation Support for the List Control Type
- UI Automation Support for the ListItem Control Type
- UI Automation Support for the Menu Control Type
- UI Automation Support for the MenuBar Control Type
- UI Automation Support for the MenuItem Control Type
- UI Automation Support for the Pane Control Type
- UI Automation Support for the ProgressBar Control Type
- UI Automation Support for the RadioButton Control Type
- UI Automation Support for the ScrollBar Control Type
- UI Automation Support for the Separator Control Type
- UI Automation Support for the Slider Control Type
- UI Automation Support for the Spinner Control Type
- UI Automation Support for the SplitButton Control Type
- UI Automation Support for the StatusBar Control Type
- UI Automation Support for the Tab Control Type
- UI Automation Support for the TabItem Control Type
- UI Automation Support for the Table Control Type
- UI Automation Support for the Text Control Type
- UI Automation Support for the Thumb Control Type
- UI Automation Support for the TitleBar Control Type
- UI Automation Support for the ToolBar Control Type
- UI Automation Support for the ToolTip Control Type
- UI Automation Support for the Tree Control Type
- UI Automation Support for the TreeItem Control Type
- UI Automation Support for the Window Control Type

See also

- ControlType

UI Automation Support for the Button Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Button control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, control patterns, and UI Automation events.

A button is an object that a user interacts with to perform an action such as the **OK** and **Cancel** buttons on a dialog box. The button control is a simple control to expose because it maps to a single command that the user wishes to complete.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Button control type. The UI Automation requirements apply to all button controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to button controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Button - Image (0 or more) - Text (0 or more)	Button

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the controls that implement the Button control type (such as button controls). For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AcceleratorKeyProperty	See notes.	The Button control typically must support an accelerator key to enable an end user to perform the action it represents quickly from the keyboard.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
ControlTypeProperty	Button	This value is the same for all UI frameworks.
HelpTextProperty	See notes.	The Help Text can indicate what the end result of activating the button will be. This is typically the same type of information presented through a ToolTip.
IsContentElementProperty	True	The Button control must always be content.
IsControlElementProperty	True	The Button control must always be a control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
LabeledByProperty	`Null`	Button controls are self-labeled by their contents.
LocalizedControlTypeProperty	"button"	Localized string corresponding to the Button control type.
NameProperty	See notes.	The name of the button control is the text that is used to label it. Whenever an image is used to label a button, alternate text must be supplied for the button's Name property.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all button controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES

CONTROL PATTERN	SUPPORT	NOTES
IInvokeProvider	See notes.	All buttons should support the Invoke control pattern or the Toggle control pattern. Invoke is supported when the button performs a command at the request of the user. This command maps to a single operation such as Cut, Copy, Paste, or Delete.
IToggleProvider	See notes.	All buttons should support the Invoke control pattern or the Toggle control pattern. Toggle is supported if the button can be cycled through a series of up to three states. Typically this is seen as an on/off switch for specific features.
IExpandCollapseProvider	See notes.	When a button is hosted as a child of a split button, the child button can support the ExpandCollapse pattern instead of the Invoke or Toggle pattern. The ExpandCollapse pattern can be used for opening or closing a menu or other sub-structure associated with the button element.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all button controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
NameProperty property-changed event.	Required	None
StructureChangedEvent	Required	None
InvokedEvent	Depends	If the control supports the Invoke control pattern, it must support this event.
ToggleStateProperty property-changed event.	Depends	If the control supports the Toggle control pattern, it must support this event.

See also

- Button
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Calendar Control Type

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Calendar control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, control patterns, and UI Automation events.

Calendar controls allow a user to easily determine the date and select other dates.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Calendar control type. The UI Automation requirements apply to all calendar controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to calendar controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Calendar • DataGrid ◦ Header (0 or 1) ◦ HeaderItem (0 or 7; quantity depends on how many days are displayed in columns) ◦ ListItem (quantity depends on how many days are displayed) ◦ Button (0 or 2; for paging calendar view)	Calendar - ListItem (quantity depends on how many days are displayed)

Calendar controls can be represented in many different forms within the user interface. The only guaranteed controls to be in the control view of the UI Automation tree are the data grid, header, header item, and list item controls.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the calendar controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
ControlTypeProperty	Calendar	This value is the same for all UI frameworks.
IsContentElementProperty	True	The calendar control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The calendar control is always included in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
LabeledByProperty	See notes.	The label of the document control. Typically, the title of the document is used.
LocalizedControlTypeProperty	"calendar"	Localized string corresponding to the Calendar control type.
NameProperty	See notes.	The calendar control typically gets its name from the current day's date.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all calendar controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT	NOTES
IGridProvider	Yes	The calendar control always supports the Grid pattern because the days within a month are items that can be navigated spatially.
IScrollProvider	Depends	Most calendar controls support flipping the view by page. The Scroll pattern is recommended in order to support paging navigation.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT	NOTES
ISelectionProvider	Depends	Most calendar controls retain a specific day, month, or year as a selection of the sub-element. Some calendars are multi-selectable and others only single-selectable.
ITableProvider	Yes	The calendar control always has a header within its subtree for the days of the week, so the Table pattern must be supported.
IValueProvider	No	The Value control pattern is not necessary for calendar controls because you cannot set the value directly on the control. If a specific date is associated with the control, the information should be provided by the Selection control pattern.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all calendar controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
LayoutInvalidatedEvent	Required	None
StructureChangedEvent	Required	None
CurrentViewProperty property-changed event.	Depends	None
HorizontallyScrollableProperty property-changed event.	Depends	If the control supports the Scroll control pattern, it must support this event.
HorizontalScrollPercentProperty property-changed event.	Depends	If the control supports the Scroll control pattern, it must support this event.
HorizontalViewSizeProperty property-changed event.	Depends	If the control supports the Scroll control pattern, it must support this event.

UI AUTOMATION EVENT	SUPPORT	NOTES
VerticalScrollPercentProperty property-changed event.	Depends	If the control supports the Scroll control pattern, it must support this event.
VerticallyScrollableProperty property-changed event.	Depends	If the control supports the Scroll control pattern, it must support this event.
VerticalViewSizeProperty property-changed event.	Depends	If the control supports the Scroll control pattern, it must support this event.
InvalidatedEvent	Required	None

See also

- Calendar
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the CheckBox Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about Microsoft UI Automation support for the CheckBox control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

A check box is an object used to indicate a state that users can interact with to cycle through that state. Check boxes either present a binary (Yes/No), (On/Off), or tertiary (On, Off, Indeterminate) option to the user.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the CheckBox control type. The UI Automation requirements apply to all check box controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to check box controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
CheckBox	CheckBox

NOTE

Check boxes never have child elements in the control or content view. If the control does need to contain child elements this indicates that another control type should be used.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to check box controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
ControlTypeProperty	CheckBox	This value is the same for all UI frameworks.
IsContentElementProperty	True	The value of this property must always be True. This means that the check box control must always be included in the content view of the UI Automation tree.
IsControlElementProperty	True	The value of this property must always be True. This means that the check box control must always be included in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
LabeledByProperty	`Null`	Check boxes are self-labeling controls.
LocalizedControlTypeProperty	"check box"	Localized string corresponding to the CheckBox control type.
NameProperty	See notes.	The value of the check box control's `Name` property is the text that is displayed beside the box that maintains the toggle state.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all check box controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IToggleProvider	Required	Allows the check box to be cycled through its internal states programmatically.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all check box controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
StructureChangedEvent	Required	None
ToggleStateProperty property-changed event.	Required	None

Default Action

The default action of the check box is to cause a radio button to become focused and toggle its current state. As mentioned previously, check boxes either present a binary (Yes/No) (On/Off) decision to the user or a tertiary (On, Off, Indeterminate). If the check box is binary the default action causes the "on" state to become "off" or the "off" state to become "on". In a tertiary state check box the default action cycles through the states of the check box in the same order as if the user had sent successive mouse clicks to the control.

See also

- CheckBox
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the ComboBox Control Type

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the ComboBox control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, control patterns, and UI Automation events.

A combo box is a list box combined with a static control or an edit control that displays the currently selected item in the list box portion of the combo box. The list box portion of the control is displayed at all times or only appears when the user selects the drop-down arrow (which is a push button) next to the control. If the selection field is an edit control, the user can enter information that is not in the list; otherwise, the user can only select items in the list.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the ComboBox control type. The UI Automation requirements apply to all combo box controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to combo box controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
ComboBox - Edit (0 or 1) - List (1) - List Item (child of List; 0 to many) - Button (1)	ComboBox - List Item (0 to many)

The edit control in the control view of the combo box is necessary only if the combo box can be edited to take any input, as is the case of the combo box in the Run dialog box.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to combo box controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
ControlTypeProperty	ComboBox	This value is the same for all UI frameworks.
HelpTextProperty	See notes.	The help text for combo box controls should explain why the user is being asked to choose an option from the combo box. The text is similar to information presented through a tooltip. For example, "Select an item to set the display resolution of your monitor."
IsContentElementProperty	True	Combo box controls are always included in the content view of the UI Automation tree.
IsControlElementProperty	True	Combo box controls are always included in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	True	Combo box controls expose a set of items from a selection container. The combo box control can receive keyboard focus, although when a UI Automation client sets focus on a combo box, any items in the combo box subtree might receive the focus.
LabeledByProperty	See notes.	Combo box controls typically have a static text label that this property references.
LocalizedControlTypeProperty	"combo box"	Localized string corresponding to the ComboBox control type.
NameProperty	See notes.	The combo box control typically gets its name from a static text control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all combo box controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IExpandCollapseProvider	Yes	The combo box control must always contain the drop-down button in order to be a combo box.
ISelectionProvider	Yes	Displays the current selection in the combo box. This support is delegated to the list box beneath the combo box.
IValueProvider	Depends	If the combo box has the ability to take arbitrary text values, the Value pattern must be supported. This pattern provides the ability to programmatically set the string contents of the combo box. If the Value pattern is not supported, this indicates that the user must make a selection from the list items within the subtree of the combo box.
IScrollProvider	Never	The Scroll pattern is never supported on a combo box directly. It is supported if a list box contained within a combo box can scroll. It may only be supported when the list box is visible on the screen.

Required Events

The following table lists the UI Automation events required to be supported by all combo box controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
StructureChangedEvent	Required	None
ExpandCollapseStateProperty property-changed event.	Required	None
ValueProperty property-changed event.	Depends	If the control supports the Value pattern, it must support this event.

See also

- ComboBox
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the DataGrid Control Type

3/12/2020 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about Microsoft UI Automation support for the DataGrid control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the `ControlType` property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

The DataGrid control type lets a user easily work with items that contain metadata represented in columns. Data grid controls have rows of items and columns of information about those items. A List View control in Microsoft Vista Explorer is an example that supports the DataGrid control type.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the DataGrid control type. The UI Automation requirements apply to all data grid controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to data grid controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

UI AUTOMATION TREE - CONTROL VIEW	UI AUTOMATION TREE - CONTENT VIEW
DataGrid • Header (0, 1, or 2) ◦ HeaderItem (number of columns or rows) • DataItem (0 or more; can be structured in hierarchy)	DataGrid - DataItem (0 or more; can be structured in hierarchy)

Required UI Automation Properties

The following table lists the properties whose value or definition is especially relevant to data grid controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
ControlTypeProperty	DataGrid	This value is the same for all UI frameworks.
IsContentElementProperty	True	The value of this property must always be True. This means that the data grid control must always be in the content view of the UI Automation tree.
IsControlElementProperty	True	The value of this property must always be True. This means that the data grid control must always be in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
LabeledByProperty	See notes.	If there is a static text label then this property must expose a reference to that control.
LocalizedControlTypeProperty	"data grid"	Localized string corresponding to the DataGrid control type.
NameProperty	See notes.	The data grid control typically gets the value for its `Name` property from a static text label. If there is not a static text label an application developer must assign a value to for the `Name` property. The value of the `Name` property must never be the textual contents of the edit control.

Required UI Automation Control Patterns

The following table lists the control patterns required to be supported by all data grid controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IGridProvider	Yes	The data grid control itself always supports the Grid control pattern because the items that it contains metadata that is laid out in a grid.

CONTROL PATTERN	SUPPORT	NOTES
IScrollProvider	Depends	The ability to scroll the data grid depends on content and whether scroll bars are present.
ISelectionProvider	Depends	The ability to select the data grid depends on content.
ITableProvider	Yes	The data grid control always has a header within its subtree so the Table control pattern must be supported.

Data items within the data grid containers will support at a minimum:

- Selection Item control pattern (if the data grid is selectable)
- Scroll Item control pattern (if the data grid is scrollable)
- Grid Item control pattern
- Table Item control pattern

Required UI Automation Events

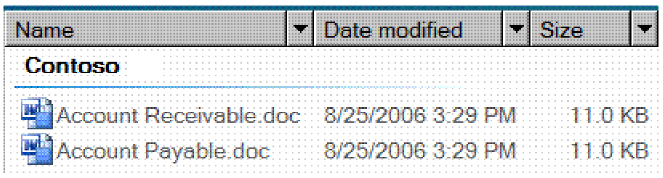
The following table lists the UI Automation events required to be supported by all data grid controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
LayoutInvalidatedEvent	Depends	None
StructureChangedEvent	Required	None
CurrentViewProperty property-changed event.	Depends	None
HorizontallyScrollableProperty property-changed event.	Depends	If the control supports the Scroll pattern, it must support this event.
HorizontalScrollPercentProperty property-changed event.	Depends	If the control supports the Scroll pattern, it must support this event.
HorizontalViewSizeProperty property-changed event.	Depends	If the control supports the Scroll pattern, it must support this event.

UI AUTOMATION EVENT	SUPPORT	NOTES
VerticalScrollPercentProperty property-changed event.	Depends	If the control supports the Scroll pattern, it must support this event.
VerticallyScrollableProperty property-changed event.	Depends	If the control supports the Scroll pattern, it must support this event.
VerticalViewSizeProperty property-changed event.	Depends	If the control supports the Scroll pattern, it must support this event.
InvalidatedEvent	Required	None

Date Grid Control Type Example

The following image illustrates a List View control that implements the DataGrid control type.



Name	Date modified	Size
Contoso		
Account Receivable.doc	8/25/2006 3:29 PM	11.0 KB
Account Payable.doc	8/25/2006 3:29 PM	11.0 KB

The control view and the content view of the UI Automation tree that pertains to the List View control is displayed below. The control patterns for each automation element are shown in parentheses.

UI AUTOMATION TREE - CONTROL VIEW	UI AUTOMATION TREE - CONTENT VIEW
• DataGrid (Table, Grid, Selection) • Header ○ HeaderItem "Name" (Invoke) ○ HeaderItem "Date Modified" (Invoke) ○ HeaderItem "Size" (Invoke) • Group "Contoso" (TableItem, GridItem, SelectionItem, Table*, Grid*) ○ DataItem "Accounts Receivable.doc" (SelectionItem, Invoke, TableItem*, GridItem*) ○ DataItem "Accounts Payable.doc" (SelectionItem, Invoke, TableItem*, GridItem*)	• DataGrid (Table, Grid, Selection) • Group "Contoso" (TableItem, GridItem, SelectionItem, Table*, Grid*) ○ DataItem "Accounts Receivable.doc" (SelectionItem, Invoke, TableItem*, GridItem*) ○ DataItem "Accounts Payable.doc" (SelectionItem, Invoke, TableItem*, GridItem*)

* The previous example shows a DataGrid that contains multiple levels of controls. The Group ("Contoso") control contains two DataItem controls ("Accounts Receivable.doc" and "Accounts Payable.doc"). A DataGrid/GridItem pair is independent of a pair at another level. The DataItem controls under a Group can also be exposed as a ListItem control type, enabling them to be presented more clearly as selectable objects, rather than as simple data elements. This example does not include the sub-elements of the grouped data items.

See also

- DataGrid
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the DataItem Control Type

1/28/2020 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about Microsoft UI Automation support for the DataItem control type. In UI Automation a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

An entry in a Contacts list is an example of a data item control. A data item control contains information that is of interest to an end user. It is more complicated than the simple list item because it contains richer information.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the DataItem control type. The UI Automation requirements apply to all data item controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to data item controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

UI AUTOMATION TREE - CONTROL VIEW	UI AUTOMATION TREE - CONTENT VIEW
DataItem - Varies (0 or more; can be structured in hierarchy)	DataItem - Varies (0 or more; can be structured in hierarchy)

A data item element in a data grid can host a variety of objects, including another layer of data items, or specific grid elements such as text, images, or edit controls. If the data item element has a specific object role, the element should be exposed as a specific control type; for example, a ListItem control type for a selectable data item in the grid.

Required UI Automation Properties

The following table lists the properties whose value or definition is especially relevant to data item controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
ControlTypeProperty	DataItem	This value is the same for all UI frameworks.
IsContentElementProperty	True	The data item control must always be content.
IsControlElementProperty	True	The data item control must always be a control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
ItemStatusProperty	See notes.	If the control contains status that is being updated dynamically then this property must be supported so that an assistive technology can receive updates when the status of the element changes.
ItemTypeProperty	See notes.	This is the string value that conveys to the end user the underlying object that the item represents. Examples are "Media File" or "Contact".
LabeledByProperty	`Null`	Data item controls do not have a static text label.
LocalizedControlTypeProperty	"data item"	Localized string corresponding to the DataItem control type.
NameProperty	See notes.	The data item control always contains a primary text element that relates to what the user would associate as the most semantic identifier for the item.

Required UI Automation Control Patterns

The following table lists the Microsoft UI Automation control patterns required to be supported by all data item controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES

CONTROL PATTERN	SUPPORT	NOTES
IExpandCollapseProvider	Depends	If the data item can be expanded or collapsed to show and hide information, the Expand Collapse pattern must be supported.
IGridItemProvider	Depends	Data items will support the Grid Item pattern when a collection of data items is available within a container that can be spatially navigated item-to-item.
IScrollItemProvider	Depends	All data items support the ability to be scrolled into view with the Scroll Item pattern when their data container has more items than can fit on the screen.
ISelectionItemProvider	Yes	All data items must support the Selection Item pattern to indicate when the item is selected.
ITableItemProvider	Depends	If the data item is contained within a Data Grid control type then it will support this pattern.
IToggleProvider	Depends	If the data item contains a state that can be cycled through.
IValueProvider	Depends	If the data item's primary text is editable then the Value pattern must be supported.

Working with Data Items in Large Lists

Large lists are often data virtualized within UI frameworks to assist in performance. Due to this, a UI Automation client cannot use the UI Automation query feature to scrape the contents of the full tree in the same way that it can in other item containers. A client should scroll the item into view (or expand the control to show all valuable options)prior to accessing the full set of information from the data item.

When calling `SetFocus` on the UI Automation element for the data item, the Microsoft Windows Explorer case will return successfully and cause focus to be set to the Edit within the data item subtree.

Required UI Automation Events

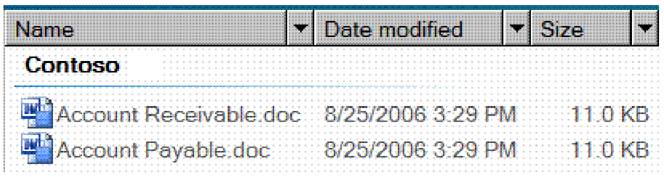
The following table lists the UI Automation events required to be supported by all data item controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None

UI AUTOMATION EVENT	SUPPORT	NOTES
IsOffscreenProperty property-changed event.	Required	None
NameProperty property-changed event.	Required	None
StructureChangedEvent	Required	None
InvokedEvent	Depends	None
ExpandCollapseStateProperty property-changed event.	Depends	None
ElementAddedToSelectionEvent	Required	None
ElementRemovedFromSelectionEvent	Required	None
ElementSelectedEvent	Required	None
ToggleStateProperty property-changed event.	Depends	None
ValueProperty property-changed event.	Depends	None

DataItem Control Type Example

The following image illustrates a DataItem control type in a List View control with support for rich information for the columns.



Name	Date modified	Size
Contoso		
Account Receivable.doc	8/25/2006 3:29 PM	11.0 KB
Account Payable.doc	8/25/2006 3:29 PM	11.0 KB

The Control View and the Content View of the UI Automation tree that pertains to the data item control is displayed below. The control patterns for each automation element are shown in parentheses. The Group "Contoso" is also part of the grid of the Data Grid host control.

UI AUTOMATION TREE - CONTROL VIEW	UI AUTOMATION TREE - CONTENT VIEW
- Group "Contoso" (Table, Grid) - DataItem "Accounts Receivable.doc" (TableItem, GridItem, SelectionItem, Invoke) - Image "Accounts Receivable.doc" - Edit "Name" (TableItem, GridItem, Value "Accounts Receivable.doc") - Edit "Date modified" (TableItem, GridItem, Value "8/25/2006 3:29 PM") - Edit "Size" (GridItem, TableItem, Value "11.0 KB) - DataItem "Accounts Payable.doc" (TableItem, GridItem, SelectionItem, Invoke) - ...	- Group "Contoso" (Table, Grid) - DataItem "Accounts Receivable.doc" (TableItem, GridItem, SelectionItem, Invoke) - Image "Accounts Receivable.doc" - Edit "Name" (TableItem, GridItem, Value "Accounts Receivable.doc") - Edit "Date modified" (TableItem, GridItem, Value "8/25/2006 3:29 PM") - Edit "Size" (GridItem, TableItem, Value "11.0 KB) - DataItem "Accounts Payable.doc" (TableItem, GridItem, SelectionItem, Invoke) - ...

If a grid represents a list of selectable items, the corresponding UI elements can be exposed with the ListItem control type instead of the DataItem control type. In the preceding example, the DataItem elements ("Accounts Receivable.doc" and "Accounts Payable.doc") under Group ("Contoso") can be improved by exposing them as ListItem control types because that type already supports the SelectionItem control pattern.

See also

- DataItem
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Document Control Type

1/28/2020 • 3 minutes to read • Edit Online

NOTE
This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Document control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, and control patterns.

Document controls let a user view and manipulate multiple pages of text. Unlike edit controls which only support a simple line of unformatted text, document controls can host text that is richly styled and formatted.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Document control type. The UI Automation requirements apply to all document controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to document controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Document - Varies	Document - Varies

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to document controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	The document has a clickable point that will cause the document of one of its elements in the document container to have focus.
ControlTypeProperty	Document	This value is the same for all UI frameworks.
IsContentElementProperty	True	The document control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The document control is always included in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
LabeledByProperty	See notes.	The value of this property should be the label of the document control. Typically, the title of the document is used.
LocalizedControlTypeProperty	"document"	Localized string corresponding to the Document control type.
NameProperty	See notes.	The document control typically gets its names from the file name it is loaded from. This is often displayed in a containing window or frame title.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by document controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IScrollProvider	Depends	The document control can span larger than that span of the viewport. The control should support the Scroll control pattern if the content is scrollable.
ITextProvider	Required	The document control can span larger than that span of the viewport. The control should support the Scroll control pattern if the content is scrollable.

CONTROL PATTERN	SUPPORT	NOTES
IValueProvider	Never	The document control does not support this control pattern because the contents of the control often span more than one page. UI Automation clients should use TextPattern to obtain text information about a document.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all document controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
StructureChangedEvent	Required	None
HorizontallyScrollableProperty property-changed event.	Required	None
HorizontalScrollPercentProperty property-changed event.	Required	None
HorizontalViewSizeProperty property-changed event.	Required	None
VerticalScrollPercentProperty property-changed event.	Required	None
VerticallyScrollableProperty property-changed event.	Required	None
VerticalViewSizeProperty property-changed event.	Required	None
InvalidatedEvent	Depends	If the control supports the Selection control pattern, it must support this event.
TextSelectionChangedEvent	Required	None
TextChangedEvent	Required	None

UI AUTOMATION EVENT	SUPPORT	NOTES
ValueProperty property-changed event.	Never	None

See also

- Document
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Edit Control Type

1/28/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Edit control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, and control patterns.

Edit controls enable a user to view and edit a simple line of text without rich formatting support.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Edit control type. The UI Automation requirements apply to all edit controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to edit controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Edit	Edit

The controls that implement the Edit control type will always have zero scroll bars in the control view of the UI Automation tree because it is a single-line control. The single line of text may wrap in some layout scenarios. The Edit control type is best suited for holding small amounts of editable or selectable text.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to edit controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	The edit control must have a clickable point that gives input focus to the edit portion of the control when a user clicks the mouse there.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The name of the edit control is typically generated from a static text label. If there is not a static text label, a property value for `Name` must be assigned by the application developer. The `Name` property should never contain the textual contents of the edit control.
LabeledByProperty	See notes.	If there is a static text label associated with the control, then this property must expose a reference to that control. If the text control is a subcomponent of another control, it will not have a `LabeledBy` property set.
ControlTypeProperty	Edit	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"edit"	Localized string corresponding to the Edit control type.
IsContentElementProperty	True	The edit control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The edit control is always included in the control view of the UI Automation tree.
IsPasswordProperty	See notes.	Must be set to true on edit controls that contain passwords. If an edit control does contain Password contents then this property can be used by a screen reader to determine whether keystrokes should be read out as the user types them.

Required UI Automation Control Patterns and Properties

The following table lists the control patterns required to be supported by all edit controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/CONTROL PATTERN PROPERTY	SUPPORT/VALUE	NOTES

CONTROL PATTERN/CONTROL PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ITextProvider	Depends	Edit controls should support the Text control pattern because detailed text information should always be available for clients.
IValueProvider	Depends	All edit controls that take a string must expose the Value pattern.
IsReadOnly	See notes.	This property must be set to indicate whether the control can have a value set programmatically or is editable by the user.
Value	See notes.	This property will return the textual contents of the edit control. If the `IsPasswordProperty` is set to `true`, this property must raise an `InvalidOperationException` when requested.
IRangeValueProvider	Depends	All edit controls that take a numeric range must expose Range Value control pattern.
Minimum	See notes.	This property must be the smallest value that the edit control's contents can be set to.
Maximum	See notes.	This property must be the largest value that the edit control's contents can be set to.
SmallChange	See notes.	This property must indicate the number of decimal places that the value can be set to. If the edit only take integers, the `SmallChangeProperty` must be 1. If the edit takes a range from 1.0 to 2.0, then the `SmallChangeProperty` must be 0.1. If the edit control takes a range from 1.00 to 2.00 then the `SmallChangeProperty` must be 0.001.
LargeChange	`Null`	This property does not need to be exposed on an edit control.
Value	See notes.	This property will indicate the numeric contents of the edit control. When a more precise value is set by a UI Automation client within the ranges specified in the `Minimum` and `Maximum` properties, the Value property will automatically be rounded to the closest accepted value.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all edit controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvalidatedEvent	Required	None
TextSelectionChangedEvent	Required	None
TextChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
NameProperty property-changed event.	Required	None
ValueProperty property-changed event.	Depends	None
HorizontallyScrollableProperty property-changed event.	Never	None
HorizontalScrollPercentProperty property-changed event.	Never	None
HorizontalViewSizeProperty property-changed event.	Never	None
VerticalScrollPercentProperty property-changed event.	Never	None
VerticallyScrollableProperty property-changed event.	Never	None
VerticalViewSizeProperty property-changed event.	Never	None
ValueProperty property-changed event.	Depends	If the control supports the range Value control pattern, it must support this event.
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Edit
- UI Automation Control Types Overview

- UI Automation Overview

UI Automation Support for the Group Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Group control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, and UI Automation control patterns.

The group control represents a node within a hierarchy. The Group control type creates a separation in the UI Automation tree so items that are grouped together have a logical division within the UI Automation tree.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Group control type. The UI Automation requirements apply to all group controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to group controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Group - 0 or many controls	Group - 0 or many controls

Typically group controls will have the UI Automation Support for the ListItem Control Type, UI Automation Support for the TreeItem Control Type, or UI Automation Support for the DataItem Control Type control types found underneath them in the subtree. Because 'Group' is a generic container, it is possible for any type of control to be under the Group control in the tree.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to group controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The group control typically gets its name from the text that labels the control.
LabeledByProperty	See notes.	Group controls are typically self-labeling. In these cases return `null` here. If there is a static text label for the group then that must be returned as the value of the LabeledBy property.
ControlTypeProperty	Group	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"group"	Localized string corresponding to the Group control type.
IsContentElementProperty	True	The group control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The calendar group is always included in the control view of the UI Automation tree.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported for the Group control type. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IExpandCollapseProvider	Depends	Group controls that can be used to show or hide information must support the Expand Collapse pattern.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all group controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
ExpandCollapseStateProperty property-changed event.	Depends	None
ToggleStateProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Group
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Header Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Header control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

The header control provides a visual container for the labels for rows or columns of information.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Header control type. The UI Automation requirements apply to all header controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to header controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Header - HeaderItem (1 or more)	None

Header controls always have 1 or more children in the control view of the UI Automation tree.

Header controls have zero children in the content view of the UI Automation tree.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to header controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The header control needs a name if there is more than one row header or more than one column header. This identifies the information within the header.
LabeledByProperty	`Null`.	Header controls do not have a static label.
ControlTypeProperty	Header	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"header"	This value is the same for all UI frameworks.
OrientationProperty	Horizontal	The value of this property exposes the position of the header control - whether it is a row header or column header.
IsContentElementProperty	False	The header control is not included in the content view of the UI Automation tree.
IsControlElementProperty	True	The header control is always included in the control view of the UI Automation tree.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all header controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
ITransformProvider	Depends	Implement this control pattern if the header control can be resized.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all header controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Header
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the HeaderItem Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the HeaderItem control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

The HeaderItem control type provides a visual label for a row or column of information.

Header item controls are examples of controls that implement the HeaderItem control type. The UI Automation requirements in the following sections apply to all header controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to header item controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
HeaderItem	None

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to header item controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The header item control is always self-labeling.
LabeledByProperty	`Null`.	Header item controls do not have a static label.
ControlTypeProperty	HeaderItem	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"header item"	Localized string for the HeaderItem control type.
IsContentElementProperty	False	The header item control is not included in the content view of the UI Automation tree.
IsControlElementProperty	True	The header item control is always included in the control view of the UI Automation tree.
ItemStatusProperty	See notes.	This property provides information for sort orders by the header item.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all header item controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
ITransformProvider	Depends	Implement this control pattern if the header item control can be resized.
IInvokeProvider	Depends	Implement this control pattern if the header item control can be clicked to sort the data.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all header item controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvokedEvent	Depends	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- HeaderItem
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Hyperlink Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Hyperlink control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Hyperlink controls enable a user to navigate within a page, from one page to another page, and open windows.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Hyperlink control type. The UI Automation requirements apply to all hyperlink controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to hyperlinks controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Hyperlink	Hyperlink

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the Hyperlink control type. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The hyperlink control's name is the text that is displayed on the screen as underlined.
ClickablePointProperty	See notes.	The hyperlink control's clickable point must be a point that launches the hyperlink if clicked with a mouse pointer.
LabeledByProperty	See notes.	If there is a static text label then this property must expose a reference to that control.
ControlTypeProperty	Hyperlink	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"hyperlink"	Localized string corresponding to the Hyperlink control type.
IsContentElementProperty	True	The hyperlink control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The hyperlink control is always included in the control view of the UI Automation tree.

Required UI Automation Control Patterns and Properties

The following table lists the UI Automation control patterns required to be supported by all hyperlink controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
IInvokeProvider	Yes	All hyperlink controls must support the Invoke pattern.
IValueProvider	Depends	Hyperlink controls should support the Value control pattern when the link contains information that is usable and meaningful to the user.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
Value	For example, `"https://www...."`	A URL for an Internet or Intranet address is an example of a hyperlink that contains information that is meaningful to the user. A programmatic link, however, is meaningful only to an application and is not recommended for the Value property.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all hyperlink controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvokedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Hyperlink
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Image Control Type

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Image control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Image controls used as icons, informational graphics, and charts will support the Image control type. Controls used as background or watermark images will not support the Image control type.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Image control type. The UI Automation requirements apply to all image controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to image controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Image	Image (Depends whether the image contains information (based on the value of `IsContentElement` property))

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the Image control type. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	The image control's clickable point must be a point within the bounding rectangle of the image control.

UI AUTOMATION PROPERTY	VALUE	NOTES
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The Name property must be exposed for all image controls that contain information. Programmatic access to this information requires that a textual equivalent to the graphic be provided. If the image control is purely decorative, it must only show up in the control view of the UI Automation tree and is not required to have a name. UI frameworks must support an ALT or alternate text property on images that can be set from within their framework. This property will then map to the UI Automation Name property.
LabeledByProperty	See notes.	If there is a static text label then this property must expose a reference to that control.
ControlTypeProperty	Image	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"image"	Localized string corresponding to the Image control type.
IsContentElementProperty	See notes.	The image control must be included in the content view of the UI Automation tree when it contains meaningful information not already exposed to the end user.
IsControlElementProperty	True	The image control is always included in the control view of the UI Automation tree.
HelpTextProperty	See notes.	The HelpText property exposes a localized string which describes the actual visual appearance of the control (for example, a red square with a white 'X') or other tooltip information associated with the image. This property must be supported when a long description is needed to convey more information about the image control. For example, a complicated chart or diagram. This property maps to the HTML LongDesc tag and the Scalable Vector Graphics (SVG) Desc tag. Developers working with image controls must support a property to allow the visual description to be set on the control. This property must be mapped to the UI Automation VisualDescription property.

UI AUTOMATION PROPERTY	VALUE	NOTES
ItemStatusProperty	See notes.	If the image control represents state information about a particular item on the screen, the control should be contained within the item. When the image is contained within an item the item must support the status property and raise appropriate notifications when the status changes. If an image is a standalone control and is conveying status this property must be supported.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all image controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IGridItemProvider	Depends	The image control supports the Grid Item pattern if the control is within a grid container.
ITableItemProvider	Depends	The image control supports the Table Item pattern if the control is within a container that has header controls.
IInvokeProvider	Never	If the image control contains a clickable image, the control should support a control type that supports the Invoke pattern, such as the Button control type.
ISelectionItemProvider	Never	Image controls should not support the Selection Item pattern.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all image controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvokedEvent	Never	None
ElementAddedToSelectionEvent	Never	None
ElementRemovedFromSelectionEvent	Never	None
ElementSelectedEvent	Never	None

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
NameProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Image
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the List Control Type

3/12/2020 • 5 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the List control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

The List control type provides a way to organize a flat group or groups of items and allows a user to select one or more of those items. The List control type has a loose restriction on what types of child elements it may contain. This enables UI Automation providers to support a well-known element for selection containers.

The UI Automation requirements in the following sections apply to all controls that implement the List control type, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms. List container controls are an example of controls that implement the List control type.

Required UI Automation Tree Structure

The following table depicts the two views of the UI Automation tree that pertain to list controls and describes what can be contained in each view. The control view contains only elements that are controls, and the content view removes redundant information from the tree. For example, a text control used to label a combo box will be exposed as the `ComboBox NameProperty`. Because the text control is already exposed in this manner through the control view it is unnecessary to have it exposed twice; therefore it is removed from the content view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Contains the elements that correspond to controls.	Removes redundant information from the tree so that assistive technologies work with the smallest set of meaningful information to the end user.
List - DataItem (0 or more) - ListItem (0 or more) - Group (0 or more) - ScrollBar (0, 1 or 2)	List - DataItem (0 or more) - ListItem (0 or more) - Group (0 or more)

The control view for a control that implements the List control type (such as a list control) consists of:

- Zero or more items within the list control (items can be based on the List Item or Data Item control types).
- Zero or more group controls within a list control.
- Zero, one, or two scroll bar controls.

The content view of a control that implements the List control type (such as a list control) consists of:

- Zero or more items within the list control (items can be based on the List Item or Data Item control types).
- Zero or more groups within the list control.

A list control must not have items that have a hierarchical relationship other than being grouped together. If the items have children in the UI Automation tree, then the list container should be based on the Tree control type.

The selectable items within the list control will be available from the descendants in the UI Automation tree of the list control. All items within the list control must belong to the same selection group. The selectable items in the list should be exposed as ListItem (instead of DataItem) control types.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to list controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	If the list control has a clickable point (a point that can be clicked to cause the list to take focus), then that point must be exposed through this property. If the value of the `IsOffScreen` property is true, then the NoClickablePointException will be raised.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The value of a list control's Name property should convey the category of options that the user is being asked to select from. This property typically gets its name from a static text label. If there is not a static text label the application developer must expose a value for the Name property. The only time this property is not required for list controls is if the control is used within the subtree of another control.
LabeledByProperty	See notes.	If there is a static text label then this property must expose a reference to that control.
ControlTypeProperty	List	This value is the same for all UI frameworks.

UI AUTOMATION PROPERTY	VALUE	NOTES
LocalizedControlTypeProperty	"list"	Localized string corresponding to the List control type.
IsContentElementProperty	True	The list control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The list control is always included in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	True	If the container can accept keyboard input then this property value should be true.
HelpTextProperty	See notes.	The Help text for list controls should explain why the user is being asked to make a choice from a list of options. For example, "Selection an item from this list will set the display resolution for your monitor."

Required UI Automation Control Patterns and Properties

The following table lists the UI Automation control patterns required to be supported by list controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ISelectionProvider	Required	All controls that support the List control type must implement `ISelectionProvider` when a selection state is maintained between the items contained in the control. If the items within the container are not selectable, the Group control type must be used.
IsSelectionRequired	Depends	List controls do not always require that an item be selected.
CanSelectMultiple	Depends	List controls can be single or multiple-selection containers.
IScrollProvider	Depends	Implement this control pattern if items in the container are scrollable.
IGridProvider	Depends	Implement this pattern when grid navigation needs to be available on an item by item basis.
IMultipleViewProvider	Depends	Implement this control pattern if the control can support multiple views of the items in the container.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ITableProvider	Never	`ITableProvider` is never supported for the List control type. If the control should support this control pattern, then the control should be based on the Data Grid control type.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all list controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT/VALUE	NOTES
InvalidatedEvent	Depends	None
LayoutInvalidatedEvent	Depends	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
CurrentViewProperty property-changed event.	Depends	None
HorizontallyScrollableProperty property-changed event.	Depends	None
HorizontalScrollPercentProperty property-changed event.	Depends	None
HorizontalViewSizeProperty property-changed event.	Depends	None
VerticalScrollPercentProperty property-changed event.	Depends	None
VerticallyScrollableProperty property-changed event.	Depends	None
VerticalViewSizeProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- List
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the ListItem Control Type

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the ListItem control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

List item controls are an example of controls that implement the ListItem control type.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the ListItem control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to list item controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
ListItem - Image (0 or more) - Text (0 or more) - Edit (0 or more)	ListItem

The children of a list item control within the content view of the UI Automation tree must always be "0". If the structure of the control is such that other items are contained underneath the list item then it should follow the requirements for the UI Automation Support for the TreeItem Control Type control type.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to list item controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	This value of this property should include the area of the image and text contents of the list item.
ClickablePointProperty	Depends	If the list control has a clickable point (a point that can be clicked to cause the list to take focus) then that point must be exposed through this property. If the list control is completely covered by descendant list items it will raise a NoClickablePointException to indicate that the client must ask an item inside the list control for a clickable point.
NameProperty	See notes.	The value of a list item control's name property comes from the text contents of the item.
LabeledByProperty	See notes.	If there is a static text label then this property must expose a reference to that control.
ControlTypeProperty	ListItem	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"list item"	Localized string corresponding to the ListItem control type.
IsContentElementProperty	True	The list control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The list control is always included in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	True	If the container can accept keyboard input then this property value should be true.
HelpTextProperty	""	The Help text for list controls should explain why the user is being asked to make a choice from a list of options, which is typically the same type of information presented through a tooltip. For example, "Select an item to set the display resolution for your monitor."
ItemTypeProperty	Depends	This property should be exposed for list item controls that are representing an underlying object. These list item controls typically have an icon associated with the control that users associate with the underlying object.

UI AUTOMATION PROPERTY	VALUE	NOTES
IsOffscreenProperty	Depends	This property must return a value for whether the list item is currently scrolled into view within the parent container that implements Scroll control pattern.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by list item controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
ISelectionItemProvider	Yes	List item control must implement this control pattern. This allows list items controls to convey when they are selected.
IScrollItemProvider	Depends	If the list item is contained within a container that is scrollable then this control pattern must be implemented.
IToggleProvider	Depends	If the list item is checkable and the action does not perform a selection state change then this control pattern must be implemented.
IExpandCollapseProvider	Depends	If the item can be manipulated to show or hide information then this control pattern must be implemented.
IValueProvider	Depends	If the item can be edited then this control pattern must be implemented. Changes to the list item control will cause changes to the values of NameProperty, and Value.
IGridItemProvider	Depends	If item to item spatial navigation is supported within the list container and the container is arranged in rows and columns then the Grid Item control pattern must be implemented.
IInvokeProvider	Depends	If the item has a command that can be performed on it, separate from selection, then this pattern must be implemented. This is typically an action associated with double-clicking the list item control. Examples would be launching a document from Microsoft Windows Explorer, or playing a music file in Microsoft Windows Media Player.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all list item controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvokedEvent	Depends	None
ElementAddedToSelectionEvent	Required	None
ElementRemovedFromSelectionEvent	Required	None
ElementSelectedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
NameProperty	Required	None
ItemStatusProperty property-changed event.	Depends	None
ExpandCollapseStateProperty property-changed event.	Depends	None
ValueProperty property-changed event.	Depends	None
ToggleStateProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- ListItem
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Menu Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about Microsoft UI Automation support for the Menu control type. It describes the control's Microsoft UI Automation tree structure and provides the properties and control patterns for specific control scenarios.

A menu control allows hierarchal organization of elements associated with commands and event handlers. In a typical Microsoft Windows application, a menu bar contains several menu buttons (such as **File**, **Edit**, and **Window**), and each menu button displays a menu. A menu contains a collection of menu items (such as **New**, **Open**, and **Close**), which can be expanded to display additional menu items or to perform a specific action when clicked.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Menu control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to menu controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Menu - MenuItem (1 or many)	Not applicable (unless the menu control is a context menu that is a parent of an object that is not a menu item) - MenuItem (1 or many)

Menu controls always appear in the control view and the content view of the UI Automation tree. Menu control types should appear under the control that their information is referring to. UI Automation clients must listen for `MenuOpenedEvent` to ensure that they consistently obtain information conveyed by menu controls. Context menu controls are a special case. They appear as children of the Desktop.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the Menu control type. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
NameProperty	Not Supported	The menu control does not require a Name property to be set.

UI AUTOMATION PROPERTY	VALUE	NOTES
LabeledByProperty	`Null`	No label is anticipated with a typical menu control.
ControlTypeProperty	Menu	This value is the same for all UI frameworks.
IsContentElementProperty	False	The menu control is not included in the content view of the UI Automation tree.
IsControlElementProperty	True	The menu control is always included in the control view of the UI Automation tree.

Required UI Automation Control Patterns

There are no required control patterns for the Menu control type.

Required UI Automation Events

Menu controls must raise `MenuOpenedEvent` when they appear on the screen. The `MenuOpenedEvent` will include the text of the control. The `MenuClosedEvent` must be raised when a menu disappears from the screen.

The following table lists the UI Automation events required to be supported by all menu controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT/VALUE	NOTES
MenuOpenedEvent	Required	None
MenuClosedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Menu
- UI Automation Control Patterns Overview
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the MenuBar Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the MenuBar control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Menu bar controls are an example of controls that implement the MenuBar control type. Menu bars provide a means for users to activate commands and options contained in an application.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the MenuBar control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to menu bar controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
MenuBar - MenuItem (1 or more) - Other controls (0 or many)	MenuBar - MenuItem (1 or more) - Other controls (0 or many)

Menu bar controls can contain other controls such as edit controls and combo boxes within its structure. These additional controls correspond to the "other controls" listed above in the control and content views.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the menu bar controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The value exposed by this property must include all of the controls contained within it.

UI AUTOMATION PROPERTY	VALUE	NOTES
NameProperty	See notes.	The menu bar control does not need a name unless an application has more than one menu bar. If there is more than one menu bar in an application, then this property should be used to expose distinguishing names, such as "Formatting" or "Outlining."
LabeledByProperty	`Null`	Menu bar controls never have a label.
ControlTypeProperty	MenuBar	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"menu bar"	Localized string corresponding to the MenuBar control type.
IsContentElementProperty	True	The menu bar control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The menu bar control is always included in the control view of the UI Automation tree.
IsOffscreenProperty	See notes.	The value of this property depends on whether the control is viewable on the screen.
OrientationProperty	Depends	This property exposes whether the menu bar control is horizontal or vertical.
IsKeyboardFocusableProperty	True	Menu bar controls are keyboard-focusable because the controls they contain can take keyboard focus.
HelpTextProperty	See notes.	No scenarios for when Help text is required for a menu bar control.
AcceleratorKeyProperty	`Null`	Menu bars never have accelerator keys.
AccessKeyProperty	"ALT"	Pressing the ALT key should always bring focus to the menu bar within the application.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by menu bar controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES

CONTROL PATTERN	SUPPORT	NOTES
IExpandCollapseProvider	Depends	If the control can be expanded or collapsed, implement IExpandCollapseProvider.
IDockProvider	Depends	If the control can be docked to different parts of the screen, implement IDockProvider.
ITransformProvider	Depends	If the control can be resized, rotated or moved it must implement ITransformProvider.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all menu bar controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT/VALUE	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
ExpandCollapseStateProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- MenuBar
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the MenuItem Control Type

1/28/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about Microsoft UI Automation support for the MenuItem control type. It describes the control's Microsoft UI Automation tree structure and provides the properties and control patterns that are required for the MenuItem control type.

A menu control allows hierarchal organization of elements associated with commands and event handlers. In a typical Microsoft Windows application, a menu bar contains several menu items (such as **File**, **Edit**, and **Window**), and each menu item displays a menu. A menu contains a collection of menu items (such as **New**, **Open**, and **Close**), which can be expanded to display additional menu items or perform a specific action when clicked. A menu item can be hosted in a menu, menu bar, or tool bar.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the MenuItem control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to menu item controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW

MenuItem "Help"
 	
• Menu (sub menu of Help menu item)
    ◦ MenuItem "Help Topics"
    ◦ MenuItem "About Notepad"
	MenuItem "Help"
	
	- MenuItem "Help Topics"
	- MenuItem "About Notepad"

The control view of the menu item control has the UI Automation tree structure shown above. Note that the **Help** menu item is included to better illustrate the structure in a typical menu to submenu hierarchy.

For the content view, Menu is absent from the UI Automation tree because it does not convey meaningful information to the end user.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to menu item controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

PROPERTY	VALUE	DESCRIPTION
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The menu item control is included in the content view of the UI Automation tree and is self labeled with a name.
LabeledByProperty	`Null`	No label.
ControlTypeProperty	MenuItem	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"menu item"	Localized string corresponding to the MenuItem control type.
IsContentElementProperty	True	The menu item control is never included in the content view of the UI Automation tree.
IsControlElementProperty	True	The menu item control must always be included in the control view of the UI Automation tree.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by menu item controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN PROPERTY	SUPPORT	NOTES
IExpandCollapseProvider	Depends	If the control can be expanded or collapsed, implement IExpandCollapseProvider.
IInvokeProvider	Depends	If the control executes a single action or command, implement IInvokeProvider.
IToggleProvider	Depends	If the control represents an option that can be turned on or off, implement IToggleProvider.

CONTROL PATTERN PROPERTY	SUPPORT	NOTES
ISelectionItemProvider	Depends	If the control is used to select from a list of options among menu items, implement ISelectionItemProvider.

UI Automation Events for Menu Item

The following table lists the Microsoft UI Automation events associated with the menu item control.

EVENT	SUPPORT	EXPLANATION
InvokedEvent	Depends	Must be raised if control supports Invoke control pattern.
ToggleStateProperty property-changed event.	Depends	Must be raised if control supports Toggle control pattern.
ExpandCollapseStateProperty property-changed event.	Depends	Must be raised if control supports Expand Collapse control pattern.
ElementSelectedEvent	Depends	None.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all menu item controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT/VALUE	NOTES
InvokedEvent	Depends	None
ElementAddedToSelectionEvent	Depends	None
ElementRemovedFromSelectionEvent	Depends	None
ElementSelectedEvent	Depends	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
ExpandCollapseStateProperty property-changed event.	Depends	None
ToggleStateProperty property-changed event.	Depends	None

UI AUTOMATION EVENT	SUPPORT/VALUE	NOTES
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

Legacy Issues

Toggle Pattern will only be supported when the Win32 menu item is checked and can be programmatically determined necessary to support Toggle Pattern. Because the Win32 menu item does not expose whether it has the ability to be checked, Invoke Pattern will be supported when the menu item is not checked. An exception will be made to always support Invoke Pattern even for menu items that should only support Toggle Pattern. This is so clients do not become confused that an element that was supporting Invoke Pattern (when menu item was unchecked) no longer supports the pattern once it becomes checked.

See also

- MenuItem
- UI Automation Control Patterns Overview
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Pane Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Pane control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

The Pane control type is used to represent an object within a frame or document window. Users can navigate between pane controls and within the contents of the current pane, but cannot navigate between items in different panes. Thus, pane controls represent a level of grouping lower than windows or documents, but above individual controls. The user navigates between panes by pressing TAB, F6, or CTRL+TAB, depending on the context. No specific keyboard navigation is required by the Pane control type.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Pane control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to pane controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Pane	Pane

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to pane controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The value for this property must always be a clear, concise and meaningful title.
ClickablePointProperty	See notes.	This property exposes a clickable point of the pane control that causes the pane to become focused when it is clicked.
LabeledByProperty	See notes.	Pane controls typically do not have a static label. If there is a static text label, it should be exposed through this property.
ControlTypeProperty	Pane	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"pane"	Localized string corresponding to the Pane control type.
IsContentElementProperty	True	Pane controls are always included in the content view of the UI Automation tree.
IsControlElementProperty	True	Pane controls are always included in the control view of the UI Automation tree.
HelpTextProperty	""	The help text for pane controls should explain why the purpose of the frame and how it relates to other frames. A description is necessary if the purpose and relationship of frames is not clear from the value of the `NameProperty` . "
AccessKeyProperty	See notes.	If a specific key combination gives focus to the pane then that information should be exposed through this property.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all pane controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
ITransformProvider	Depends	Implement this control pattern if the pane control can be moved, resized, or rotated on the screen.
IWindowProvider	Never	If you need to implement this control pattern, your control should be based on the Window control type.

CONTROL PATTERN	SUPPORT	NOTES
IDockProvider	Depends	Implement this control pattern if the pane control can be docked.
IScrollProvider	Depends	Implement this control pattern if the pane control can be scrolled.

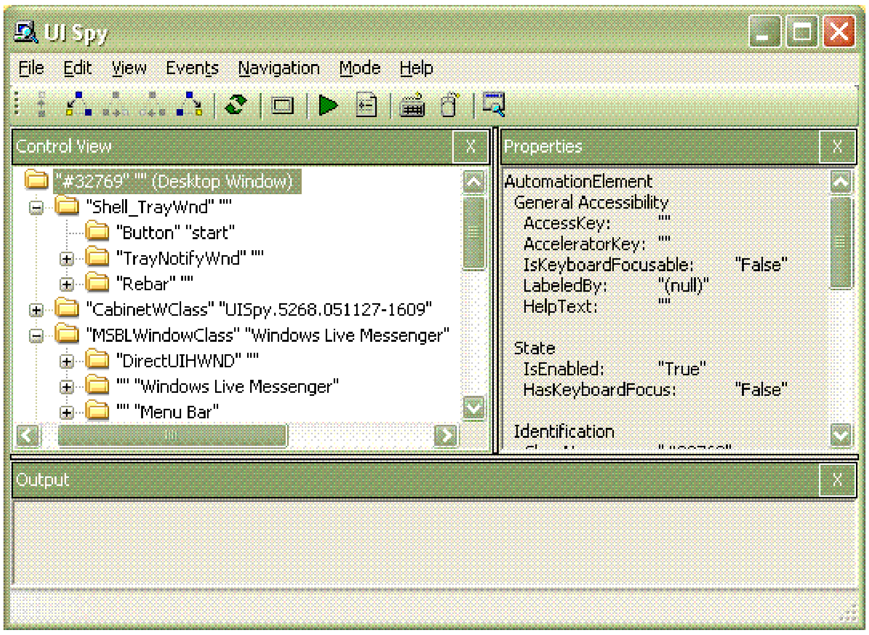
Required UI Automation Events

The following table lists the UI Automation events required to be supported by all pane controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT/VALUE	NOTES
WindowClosedEvent	Never	None
WindowOpenedEvent	Never	None
AsyncContentLoadedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
HorizontallyScrollableProperty property-changed event.	Depends	None
HorizontalScrollPercentProperty property-changed event.	Depends	None
HorizontalViewSizeProperty property-changed event.	Depends	None
VerticalScrollPercentProperty property-changed event.	Depends	None
VerticallyScrollableProperty property-changed event.	Depends	None
VerticalViewSizeProperty property-changed event.	Depends	None
WindowVisualStateProperty property-changed event.	Never	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

Pane Control Type Example

The following image illustrates a control that implements the Pane control type.



```
"#32769" "" (Desktop Window)
  "Shell_TrayWnd" ""
    "Button" "start"
    "TrayNotifyWnd" ""
    "Rebar" ""
  "CabinetWClass" "UISpy.5268.051127-1609"
  "MSBLWindowClass" "Windows Live Messenger"
    "DirectUIHWND" ""
    "" "Windows Live Messenger"
    "" "Menu Bar"
```

```
AutomationElement
 General Accessibility
  AccessKey:           ""
  AcceleratorKey:      ""
  IsKeyboardFocusable:      "False"
  LabeledBy:           "(null)"
  HelpText:            ""

 State
  IsEnabled:           "True"
  HasKeyboardFocus:         "False"

 Identification
```

UI AUTOMATION TREE - CONTROL VIEW	UI AUTOMATION TREE - CONTENT VIEW

• Pane
• Tree (Scroll Pattern)
  ◦ TreeItem
  ◦ Pane
  ◦ Edit (Scroll Pattern
	- Pane
	- Tree (Scroll Pattern)
	- TreeItem
	- ...Pane
	- Edit
	- (Scroll Pattern)

See also

- Pane
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the ProgressBar Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the ProgressBar control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, control patterns, and UI Automation events.

Progress bar controls are an example of controls that implement the ProgressBar control type. Progress bar controls are used to indicate the progress of a lengthy operation. The control consists of a rectangle that is gradually filled with the system highlight color as an operation progresses.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the ProgressBar control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to progress bar controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
ProgressBar	ProgressBar

The progress bar controls do not have any children in the control or content view of the UI Automation tree.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to progress bar controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The progress bar control typically gets its name from a static text label. If there is not a static text label the application developer must expose a value for the `Name` property.
LabeledByProperty	See notes.	If there is a static text label then this property must expose a reference to that control.
ControlTypeProperty	ProgressBar	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"progress bar"	Localized string corresponding to the ProgressBar control type.
IsContentElementProperty	True	The progress bar control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The progress bar control is always included in the control view of the UI Automation tree.

Required UI Automation Control Patterns and Properties

The following table lists the UI Automation control patterns required to be supported by progress bar controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
IValueProvider	Depends	Progress bar controls that give a textual indication of progress must implement IValueProvider.
IsReadOnly	True	The value for this property is always True.
Value	See notes.	This property exposes textual progress of a progress bar control.
IRangeValueProvider	Depends	Progress bar controls that take a numeric range must implement IRangeValueProvider

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
Minimum	0.0	The value of this property is the smallest value that the control can be set to.
Maximum	100.0	The value of this property is the largest value that the control can be set to.
SmallChange	NaN	This property is not required because progress bar controls are read-only.
LargeChange	NaN	This property is not required because progress bar controls are read-only.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all progress bar controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
NameProperty property-changed event.	Required	None
ValueProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- ProgressBar
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the RadioButton Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the RadioButton control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

A radio button consists of a round button and application-defined text (a label), an icon, or a bitmap that indicates a choice the user can make by selecting the button. An application typically uses radio buttons in a group box to permit the user to choose from a set of related, but mutually exclusive options. For example, the application might present a group of radio buttons from which the user can select a format preference for text selected in the client area. The user could select a left-aligned, right-aligned, or centered format by selecting the corresponding radio button. Typically, the user can select only one option at a time from a set of radio buttons.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the RadioButton control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to radio button controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
RadioButton	RadioButton

There are no children in the control view or the content view.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the RadioButton control type. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The radio button control's name is the text that is displayed beside the button that maintains selection state.
ClickablePointProperty	See notes.	The radio button control's clickable point MUST be a point that sets selection on the radio button if clicked with a mouse pointer.
LabeledByProperty	`Null`	Radio buttons are self-labeling controls.
ControlTypeProperty	RadioButton	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"radio button"	Localized string corresponding to the RadioButton Control Type.
IsContentElementProperty	True	The radio button control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The radio button control is always included in the control view of the UI Automation tree.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all radio button controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/CONTROL PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ISelectionItemProvider	Yes	All radio button controls must support the Selection Item pattern to enable themselves to be selected.
SelectionContainer	See notes.	The `SelectionContainerProperty` must always be completed so that a UI Automation client can determine what other radio buttons within a specific context relate to one another. For the Win32 version of the radio button, this property will not be supported because it is not possible to obtain this information from that legacy framework.

CONTROL PATTERN/CONTROL PATTERN PROPERTY	SUPPORT/VALUE	NOTES
IToggleProvider	Never	The radio button cannot cycle through its state once it has been set. This pattern must never be supported on radio button.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all radio button controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
ElementRemovedFromSelectionEvent	Required	None
ElementSelectedEvent	Required	None
ToggleStateProperty property-changed event.	Never	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- RadioButton
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the ScrollBar Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the ScrollBar control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Scroll bar controls enable a user to scroll content within a window or item container. The control is made up of a set of buttons and a thumb control.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the ScrollBar control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to scroll bar controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
ScrollBar - Button (2 or 4) - Thumb (0 or1)	Not applicable. The scroll bar control does not contain content.

The scroll bar control always has three to five children. Because the subtree has more than one button control, you must set a specific AutomationIdProperty value to each item to make them discoverable for test automation tools.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to scroll bar controls. Note that a scroll bar control never has content; its functionality is exposed through the Scroll control pattern, which is supported on the container being scrolled.

For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	`Null`	The scroll bar control does not have content elements and the `NameProperty` is not required to be set.
ClickablePointProperty	Not a number.	The scroll bar control does not have clickable points.
LabeledByProperty	`Null`	Scroll bars do not have labels.
ControlTypeProperty	ScrollBar	This value is the same for all frameworks. Scroll bars that function as sliders must use the Slider control type.
LocalizedControlTypeProperty	"scroll bar"	Localized string that corresponds to the Button control type.
IsContentElementProperty	False	The scroll bar control is never a content element. If the scroll bar is a standalone control, then it must fulfill the Slider control type and return `ControlType.Slider` for the `ControlType` property.
IsControlElementProperty	True	The scroll bar must always be a control.
OrientationProperty	True	The scroll bar control must always expose its horizontal or vertical orientation.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by scroll bar controls. For more information on control patterns, see UI Automation Control Patterns Overview. Note that when a scroll bar is used as a control for mouse manipulation only, it does not support control patterns. If it is used as a slider control within an application, it must be given the Slider control type.

CONTROL PATTERN	SUPPORT	NOTES
IScrollProvider	Never	The Scroll control pattern is never directly supported on the scroll bar.

CONTROL PATTERN	SUPPORT	NOTES
IRangeValueProvider	Depends	This functionality is required to be supported only if the Scroll control pattern is not supported on the container that has the scroll bar.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all scroll bar controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT/VALUE	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
HorizontallyScrollableProperty property-changed event.	Never	None
HorizontalScrollPercentProperty property-changed event.	Never	None
HorizontalViewSizeProperty property-changed event.	Never	None
VerticalScrollPercentProperty property-changed event.	Never	None
VerticallyScrollableProperty property-changed event.	Never	None
VerticalViewSizeProperty property-changed event.	Never	None
ValueProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- ScrollBar
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Separator Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Separator control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, and control patterns.

Separator controls are used to visually divide a space into two regions. For example, a separator control can be a bar that defines two panes in a window. If the separator can be moved, the control should be exposed as Thumb in control type.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Separator control type. The UI Automation requirements apply to all list controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the Control View and the Content View of the UI Automation tree that pertains to separator controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Separator	- The Separator control never has content.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to separator controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes	If the control can receive keyboard focus, it must support this property.
NameProperty	""	The separator control does not require a NameProperty.
LabeledByProperty	`null`	The separator control does not have a static label.
ControlTypeProperty	Separator	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"Separator"	Localized string corresponding to the Separator control type.
IsContentElementProperty	False	The separator control is never content.
IsControlElementProperty	True	The separator control must always be a control.

Required UI Automation Control Patterns

The separator control is not required to support any control patterns.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all separator controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event	Required	None
IsOffscreenProperty property-changed event	Required	None
IsEnabledProperty property-changed event	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Separator

- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Slider Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Slider control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control types.

The Slider control is a composite control with buttons that enable a user with a mouse to set a numerical range or select from a set of items.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Slider control type. The UI Automation requirements apply to all slider controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the Control View and the Content View of the UI Automation tree that pertains to slider controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Slider - Button (2 or 4) - Thumb (only 1) - List Item (0 or more)	Slider - List Item (0 or more)

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the Slider control type. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes	The majority of slider controls must raise the NoClickablePointException because the entire bounding rectangle of the slider control is occupied by child controls.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The name of the edit control is typically generated from a static text label. If there is not a static text label, a property value for `Name` must be assigned by the application developer. The `Name` property should never contain the textual contents of the edit control.
LabeledByProperty	See notes.	If there is a static text label associated with the control, then this property must expose a reference to that control. If the text control is a subcomponent of another control, it will not have a `LabeledBy` property set.
ControlTypeProperty	Slider	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"slider"	Localized string corresponding to the Edit Control Type.
IsContentElementProperty	True	The edit control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The edit control is always included in the control view of the UI Automation tree.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by all slider controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
ISelectionProvider	Depends	A slider should support the Selection control pattern if the content represents one value among a discrete set of options. When the Selection control pattern is supported, the corresponding selection must be exposed as one or more child list items of the slider.

CONTROL PATTERN	SUPPORT	NOTES
IRangeValueProvider	Depends	A slider should support the RangeValue control pattern if the content can be set to a value within a numerical range.
IValueProvider	Depends	A slider should support the Value control pattern if the content represents one value among a discrete set of options.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all slider controls.

For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvalidatedEvent	Depends	None
BoundingRectangleProperty property-changed event	Required	None
IsOffscreenProperty property-changed event	Required	None
IsEnabledProperty property-changed event	Required	None
ValueProperty property-changed event	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Slider
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Spinner Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Spinner control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Spinner controls are used to select from a domain of items or a range of numbers.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Spinner control type. The UI Automation requirements apply to all spinner controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertain to spinner controls when they support the Range Value, Value, and Selection control patterns and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

Range Value or Value control pattern

CONTROL VIEW	CONTENT VIEW
Spinner - Edit (0 or 1) - Button (2)	Spinner

Selection control pattern

CONTROL VIEW	CONTENT VIEW
Spinner - Edit (0 or 1) - Button (2) - List Item (0 or more)	Spinner - ListItem (0 or more)

To ensure that the two buttons in the control view subtree can be distinguished by automated test tools, assign the `SmallIncrement` or `SmallDecrement` `AutomationId` as appropriate. For some implementations, the associated Edit control may be a peer of the Spinner control.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to spinner controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	The spinner control's clickable point gives focus to the edit portion of the control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The spinner control typically gets its name from a static text label.
LabeledByProperty	See notes.	Spinner controls have a static text label.
ControlTypeProperty	Spinner	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"spinner"	Localized string corresponding to the Spinner control type.
IsContentElementProperty	True	The spinner control must always be content.
IsControlElementProperty	True	The Spinner control must always be a control.

Required UI Automation Control Patterns and Properties

The following table lists the UI Automation control patterns required to be supported by spinner controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ISelectionProvider	Depends	Spinner controls that have a list of items to be selected must support this pattern.
CanSelectMultiple	False	Spinner controls are always single selection containers.
IRangeValueProvider	Depends	Spinner controls that span a numeric range can support this pattern.
IValueProvider	Depends	Spinner controls that span a discrete set of options or numbers can support this pattern.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all spinner controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvalidatedEvent	Depends	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
ValueProperty property-changed event.	Depends	None
ValueProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Spinner
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the SplitButton Control Type

1/28/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the SplitButton control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

The split button control enables the ability to perform an action on a control and expand the control to see a list of other possible actions that can be performed.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the SplitButton control type. The UI Automation requirements apply to all split button controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to split button controls and describes what can be contained in each view. For more information about the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
SplitButton • Image (0 or 1) • Text (0 or 1) • Button (1 or 2) ◦ Menu (0 or 1; appears as child of button that supports ExpandCollapse pattern) ◦ MenuItem (1 to many)	SplitButton - MenuItem (1 to many)

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to split button controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	"Back"	The split button control's name is displayed on the button.
LabeledByProperty	Null	Split button controls do not have a static text label.
ControlTypeProperty	SplitButton	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"split button"	Localized string corresponding to the SplitButton control type.
HelpTextProperty	See notes.	The help text can indicate the result of activating the split button, which is typically the same type of information presented through a tooltip.
IsContentElementProperty	True	The split button control contains information for the end user.
IsControlElementProperty	True	The split button control is visible to the end user.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by split button controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IInvokeProvider	Required	Split buttons always have a default action associated with Invoke.
IExpandCollapseProvider	Required	Split buttons always have the ability to expand a list of options.

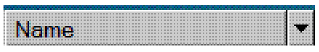
Required UI Automation Events

The following table lists the UI Automation events required to be supported by all split button controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvokedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
ExpandCollapseStateProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

SplitButton Control Example

The following image illustrates a SplitButton control type in a data grid control.



The Control View and the Content View of the UI Automation tree that pertains to the data grid and split button controls is displayed below. The control patterns for each automation element are shown in parentheses.

UI AUTOMATION TREE - CONTROL VIEW	UI AUTOMATION TREE - CONTENT VIEW

• SplitButton "Name" (Invoke, ExpandCollapse)
• Button "More options" (Invoke)
    ◦ Menu
    ◦ MenuItem
    ◦ ...
	• SplitButton "Name" (Invoke, ExpandCollapse)
	• Button "More options" (Invoke)
	    ◦ Menu
	    ◦ MenuItem
	    ◦ ...

See also

- SplitButton
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the StatusBar Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the StatusBar control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

A status bar control displays information about an object being viewed in a window of an application, the object's component, or contextual information that relates to that object's operation within your application.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the StatusBar control type. The UI Automation requirements apply to all status bar controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to status bar controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
StatusBar - Edit (0 or more) - Progress Bar (0 or many) - Image (0 or many) - Button (0 or many)	StatusBar - Edit (0 or more) - ProgressBar (0 or many) - Image (0 or many) - Button (0 or many)

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to progress bar controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The bounding rectangle of a status bar must encompass all of the controls contained within it.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The status bar control does not need a name unless more than one is used within an application. In this case, distinguish each bar with names such as "Internet Status" or "Application Status."
LabeledByProperty	`Null`	The status bar control usually does not have a label.
ControlTypeProperty	StatusBar	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"status bar"	Localized string corresponding to the StatusBar control type.
IsContentElementProperty	True	The status bar control always contains content.
IsControlElementProperty	True	The status bar control is always a control.
IsOffscreenProperty	Depends	A status bar control will return True for this property if it is not currently visible on the screen.
OrientationProperty	Depends	The value of the control's orientation: horizontal or vertical.
IsKeyboardFocusableProperty	False	Not applicable
AcceleratorKeyProperty	`Null`	Status bars do not have accelerator keys.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by status bar controls. For more information about control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES

CONTROL PATTERN	SUPPORT	NOTES
IGridProvider	Optional	Status bar controls should support the Grid control pattern so that individual pieces can be monitored and easily referenced for information.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all status bar controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- StatusBar
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Tab Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Tab control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and UI Automation. control patterns.

A tab control is analogous to the dividers in a notebook or the labels in a file cabinet. By using a tab control, an application can define multiple pages for the same area of a window or dialog box.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Tab control type. The UI Automation requirements apply to all tab controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to tab controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Tab • TabItem (1 or more) • ScrollBar (0 or 1) ◦ Button (0 or 2)	Tab - TabItem (1 or more)

Tab controls have child UI Automation elements based on the Tab Item control type. When tab items are grouped (for example, as in Microsoft Office 2007 applications) the Tab control type can also host Groups control types for the grouped tab items, as the following tree structure shows.

CONTROL VIEW	CONTENT VIEW
Tab • TabItem (1 or more) • Group (0 or more) ◦ TabItem (0 or more) • ScrollBar (0 or more) ◦ Button (0 or 2)	Tab • TabItem (1 or more) • Group (0 or more) ◦ TabItem (0 or more)

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to the Tab control type. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The tab control rarely requires a Name property.
ClickablePointProperty	No	The tab control does not have a clickable point.
LabeledByProperty	See notes.	Tab controls typically have a static text label that is exposed through this property.
ControlTypeProperty	Tab	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"tab"	Localized string corresponding to the Tab control type.
IsKeyboardFocusableProperty	True	The Tab control type must be able to receive keyboard focus. Typically, a UI Automation client calls SetFocus on a tab control and one of its items will forward the keyboard focus to the tab control. It is possible for some tab containers to take focus without setting focus to one of its items.
IsContentElementProperty	True	The tab control is always included in the content view of the UI Automation tree.

UI AUTOMATION PROPERTY	VALUE	NOTES
IsControlElementProperty	True	The tab control is always included in the control view of the UI Automation tree.
OrientationProperty	See notes.	The tab control must always indicate whether it is positioned horizontally or vertically.

Required UI Automation Control Patterns and Properties

The following table lists the UI Automation control patterns required to be supported by all tab controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ISelectionProvider	Yes	All tab controls must support the Selection pattern.
IsSelectionRequired	True	Tab controls always require that a selection be made.
CanSelectMultiple	False	Tab controls are always single-selection containers.
IScrollProvider	Depends	The Scroll pattern must be supported in the tab control has widgets that allow for a set of tab items to be scrolled through.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all tab controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
HorizontallyScrollableProperty property-changed event.	Depends	None
HorizontalScrollPercentProperty property-changed event.	Depends	None
HorizontallyScrollableProperty property-changed event.	Depends	None

UI AUTOMATION EVENT	SUPPORT	NOTES
HorizontalViewSizeProperty property-changed event.	Depends	None
VerticalScrollPercentProperty property-changed event.	Depends	None
VerticalViewSizeProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Tab
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the TabItem Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the TabItem control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

A tab item control is used as the control within a tab control that selects a specific page to be shown in a window.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the TabItem control type. The UI Automation requirements apply to all tab item controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to tab item controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
TabItem - Image (0 or 1) - Text - Pane ◦ Various controls (0 or more)	TabItem - Pane ◦ Various controls (0 or more)

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to tab item controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	The tab item control must have a clickable point that causes the item to become selected.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The tab item control is self-labeled.
LabeledByProperty	`Null`	The tab item control does not have a static text label.
ControlTypeProperty	TabItem	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"tab item"	Localized string corresponding to this control type.
IsContentElementProperty	True	The tab item control must always be content.
IsControlElementProperty	True	The tab item control must always be a control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by tab item controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
ISelectionItemProvider	Yes	The tab item control must support SelectionItemPattern.
IInvokeProvider	No	The tab item control never supports InvokePattern.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all tab item controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
ElementSelectedEvent	Required	None
ElementRemovedFromSelectionEvent	Required	None
StructureChangedEvent	Required	None

See also

- TabItem
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Table Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Table control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Table controls contain rows and columns of text, and optionally, row headers and column headers.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Table control type. The UI Automation requirements apply to all table controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to table controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Table - Header (0 or 1) - Text (0 or 1) - Various controls (0 or more)	Table - Text (0 or more) - Various controls (0 or more)

If a table control has row or column headers, they must be exposed in the Control View of the UI Automation tree. The Content View does not need to expose this information because it can be accessed using the TablePattern.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to Table controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The table control typically gets its name from a static text label. If there is no static text label, you must assign a Name property that must always be available to explain the purpose of the table.
LabeledByProperty	See notes.	If there is a static text label, this property should expose a reference to the automation element of the control.
ControlTypeProperty	Table	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"table"	Localized string corresponding to the Table control type.
HelpTextProperty	See notes.	More details about the purpose of the table should be exposed through this property if it is not sufficiently explained by accessing the NameProperty.
IsContentElementProperty	True	The table control must always be content.
IsControlElementProperty	True	The table control must always be a control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by Table controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IGridProvider	Yes	The table control always supports this control pattern because the items that it contains have data that is presented in a grid.
IGridItemProvider	Yes (required with child objects)	The inner objects of a table should support both the GridItem and TableItem control patterns. The table itself need not support the GridItem or TableItem control patterns unless the table is part of another table.

CONTROL PATTERN	SUPPORT	NOTES
ITableProvider	Yes	The table control always has the capability of having headers associated with the content.
ITableItemProvider	Yes (required with child objects)	The inner objects of a table should support both the GridItem and TableItem control patterns. The table itself need not support the GridItem or TableItem control patterns unless the table is part of another table.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all table controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Table
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Text Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Text control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Text controls are the basic user interface item that represents a piece of text on the screen.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Text control type. The UI Automation requirements apply to all text controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to text controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Text	Text (if content)

A text control can be used alone as a label or as static text on a form. It can also be contained within the structure of a:

- ListItem
- TreeItem
- DataItem

Text controls may not be in the Content View of the UI Automation tree because text is often displayed through the `NameProperty` of another control. For example the text that is used to label a Combo Box control is exposed through the control's `NameProperty` value. Because the Combo Box control is in the content view of the UI Automation Tree, it is not necessary for the text control to be there. Text controls always have 0 children in the content view

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to text controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The text bar control's name is always the txt that it displays.
LabeledByProperty	`Null`	Text controls do not have a static text label.
ControlTypeProperty	Text	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"text"	Localized string corresponding to the text control type.
IsContentElementProperty	Depends	The text control will be content if it contains information not exposed in another control's NameProperty.
IsControlElementProperty	True	The text control must always be a control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by text controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IValueProvider	Never	Text never supports ValuePattern. If the text is editable, this it is the Edit control type.
ITextProvider	Depends	Text should support the Text control pattern for better accessibility; however, it is not required. The Text control pattern is useful when the text has rich style and attributes (for example, color, bold, and italics).Depends on framework.

CONTROL PATTERN	SUPPORT	NOTES
ITableItemProvider	Depends	If the text element is contained within a Table control, this must be supported.
IRangeValueProvider	Depends	If the text element is contained within a table control, this must be supported.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all text controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
TextSelectionChangedEvent	Required	None
TextChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
NameProperty property-changed event.	Required	None
ValueProperty property-changed event.	Never	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Text
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Thumb Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Thumb control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, and control patterns.

Thumb controls provide the functionality that enables a control to be moved (or dragged), such as a scroll bar button, or resized, such as a window resizing widget. Thumb controls can also be implemented as movable borders of panes. Note that it does not provide drag-and-drop functionality. Thumb controls can receive mouse focus but usually not keyboard focus. The control developer must implement the control so that it acts appropriately (can be dragged or resized).

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Thumb control type. The UI Automation requirements apply to all thumb controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to thumb controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Thumb	- Not applicable

Thumb controls never appear in Content View because they only exist for being manipulated with a mouse. Their functionality is exposed though another control pattern, such as Scroll Pattern, Transform Pattern, or RangeValue Pattern, being supported on the Thumb container.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to thumb controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Any point within the visible client area of the Thumb control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	`Null`	The Thumb control is not available in the Content View of the UI Automation tree so it does not require a name.
LabeledByProperty	`Null`	Thumb controls never have a label.
ControlTypeProperty	Thumb	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"thumb"	Localized string corresponding to the Thumb control type.
IsContentElementProperty	False	The Thumb control is never content.
IsControlElementProperty	True	The Thumb control must always be a control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by thumb controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ITransformProvider	Required	Enables the thumb control to be moved on the screen.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all thumb controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Thumb
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the TitleBar Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the TitleBar control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Title bar controls represent titles or caption bars in a window.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the TitleBar control type. The UI Automation requirements apply to all title bar controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to title bar controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
TitleBar - Menu (0 or 1) - Button (0 or more)	Not applicable. (the title bar control has no content.)

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to TitleBar controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The bounding rectangle of a title bar must encompass all of the controls contained within it.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	False	Title bars never have keyboard focus.
NameProperty	""	The title bar is not content; its textual information is exposed on the parent window.
LabeledByProperty	See notes.	The title bar control usually does not have a label.
ControlTypeProperty	TitleBar	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"title bar"	Localized string corresponding to the TitleBar control type.
IsContentElementProperty	False	The title bar control is never content.
IsControlElementProperty	True	The title bar control must always be a control.
IsOffscreenProperty	Depends	This control will return a value depending on whether the title bar is visible on the screen.
HelpTextProperty	""	It is not necessary to expose Help text.
AcceleratorKeyProperty	""	Title bars never have accelerator keys.
AccessKeyProperty	""	The title bar control does not have an access key.

Required UI Automation Control Patterns

The TitleBar control type is not required to support any control patterns. Its functionality is exposed through the Window control pattern on the Window control.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all title bar controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None

UI AUTOMATION EVENT	SUPPORT	NOTES
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Never	None
AutomationFocusChangedEvent	Never	None
StructureChangedEvent	Required	None

See also

- TitleBar
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the ToolBar Control Type

3/12/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the ToolBar control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns. Tool bar controls enable end users to activate commands and tools contained within a application.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the ToolBar control type. The UI Automation requirements apply to all tool bar controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to tool bar controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
ToolBar - Various controls (0 or more)	ToolBar - Various controls (0 or more)

A tool bar control can contain any type of control within its subtree. They most often contain buttons, combo boxes, and split buttons.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to tool bar controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.

UI AUTOMATION PROPERTY	VALUE	NOTES
ClickablePointProperty	See notes.	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	Depends	The tool bar control does not need a name unless more than one is used within an application. If more than one is present, each must have a distinguishing name (for example, Formatting or Outlining).
LabeledByProperty	`Null`	Tool bar controls never have a label.
ControlTypeProperty	ToolBar	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"tool bar"	Localized string corresponding to the ToolBar control type.
IsContentElementProperty	True	The tool bar control is always content.
IsControlElementProperty	True	The tool bar control must always be a control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by tool bar controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
ExpandCollapsePattern	Depends	If the tool bar can be expanded and collapsed to show more items, then it must support this pattern.
DockPattern	Depends	If the tool bar can be docked to different parts of the screen, then it must support this pattern.
TransformPattern	Depends	If the tool bar can be resized, rotated, or moved, it must support this pattern.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all tool bar controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
ExpandCollapseStateProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- ToolBar
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the ToolTip Control Type

3/12/2020 • 3 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the ToolTip control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

Tool tip controls are pop-up windows that contain text.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the ToolTip control type. The UI Automation requirements apply to all tool tip controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to tool tip controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
ToolTip - Text (0 or more) - Image (0 or more)	ToolTip

Tool tip controls appear only in the Content View of the UI Automation tree if they can receive keyboard focus. Otherwise, all of the tool tip's information is available from the `HelpTextProperty` on the UI Automation element that the tool tip is referring to.

Tool tips should appear beneath the control that their information is referring to. Clients must listen for the `ToolTipOpenedEvent` to ensure that they consistently obtain information contained in tool tips.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to tool tip controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.

UI AUTOMATION PROPERTY	VALUE	NOTES
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	The clickable point should be the part of the tool tip that will dismiss the control. Some tool tips do not have this ability and will not have a clickable point.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
NameProperty	See notes.	The name of the tool tip control is the text that is displayed within the tool tip.
LabeledByProperty	`Null`	Tool tip controls are always self-labeled by their contents.
ControlTypeProperty	ToolTip	This value is the same for all UI frameworks.
LocalizedControlTypeProperty	"tool tip"	Localized string corresponding to the ToolTip control type.
IsContentElementProperty	Depends	If the tool tip control can receive keyboard focus, it must be in the Content View of the tree. If it is text only, then it is available as the HelpTextProperty from the control that raised it.
IsControlElementProperty	True	The tool tip control must always be a control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by tool tip controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IWindowProvider	Depends	Tool tips that can be closed by clicking a UI item must support WindowPattern so that they can closed automatically.
ITextProvider	Depends	For better accessibility, a tool tip control can support the Text control pattern, although it is not required. The Text control pattern is useful when the text has rich style and attributes (for example, color, bold, and italics).

Required UI Automation Events

Tool tip controls must raise the `ToolTipOpenedEvent` when they appear on the screen. The event will include a reference to the UI Automation element of the tool tip itself.

The following table lists the UI Automation events required to be supported by all tool tip controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
TextSelectionChangedEvent	Depends	None
TextChangedEvent	Depends	None
WindowClosedEvent	Depends	None
WindowOpenedEvent	Depends	None
ToolTipOpenedEvent	Required	None
ToolTipClosedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
NameProperty property-changed event.	Required	None
WindowVisualStateProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- ToolTip
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Tree Control Type

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Tree control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, and control patterns.

The Tree control type is used for containers whose contents have relevance as a hierarchy of nodes, as with the way files and folders are displayed in the left pane of Microsoft Windows Explorer. Each node has the potential to contain other nodes, called child nodes. Parent nodes, or nodes that contain child nodes, can be displayed as expanded or collapsed.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the Tree control type. The UI Automation requirements apply to all tree controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to tree controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Tree • DataItem (0 or more) • TreeItem (0 or more) ◦ TreeItem (0 or more)• ... • ScrollBar (0, 1, 2)	Tree • DataItem (0 or more) • TreeItem (0 or more) ◦ TreeItem (0 or more)• ...

The control view of the UI Automation tree consists of:

- Zero to many items within the container (items can be based on the Tree Item, Data Item, or other control type).
- Zero, one or two scroll bars.

The content view of the UI Automation tree consists of zero or many items within the container (items can be based on the Tree Item, Data Item, or other control type).

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to list controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	Tree controls have a clickable point that will cause the tree or one the items in the tree container to have the focus set on them. You get a clickable point for a tree only if you can click somewhere that doesn't cause one of the items to be selected/get focus.
ControlTypeProperty	Tree	This value is the same for all UI frameworks.
IsContentElementProperty	True	The tree control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The tree control is always included in the control view of the UI Automation tree.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
LabeledByProperty	See notes.	If the tree control has a label associated with it, this property will return an AutomationElement for that label. Otherwise, the property will return a null reference (`Nothing` in Microsoft Visual Basic .NET).
LocalizedControlTypeProperty	"tree"	Localized string corresponding to the List control type.
NameProperty	See notes.	The value of a tree control's name property usually comes from text that labels the control. If there is no text label, then the application developer must provide a value for this property.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by list controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
ISelectionProvider	Depends	Tree controls that contain a set of selectable items must implement this control pattern. This control pattern does not have to be implemented if selecting an item does not convey meaningful information to the user.
CanSelectMultiple	See notes.	Implement this property if the tree control supports multiple selection (most tree controls do not support multiple selection).
IsSelectionRequired	See notes.	The value of this property is exposed if the control requires that an item be selected.
IScrollProvider	Depends	Implement this control pattern if the contents of the tree container can be scrolled.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all tree controls. For more information on events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
InvalidatedEvent	Depends	None
BoundingRectangleProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
HorizontallyScrollableProperty property-changed event.	Depends	None
HorizontalScrollPercentProperty property-changed event.	Depends	None
HorizontalViewSizeProperty property-changed event.	Depends	None
VerticalScrollPercentProperty property-changed event.	Depends	None
VerticallyScrollableProperty property-changed event.	Depends	None

UI AUTOMATION EVENT	SUPPORT	NOTES
VerticalViewSizeProperty property-changed event.	Depends	None
AutomationFocusChangedEvent	Required	None
StructureChangedEvent	Required	None

See also

- Tree
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the TreeItem Control Type

3/12/2020 • 4 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the TreeItem control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values and control patterns.

The TreeItem control type represents a node within a tree container. Each node might contain other nodes, called child nodes. Parent nodes, or nodes that contain child nodes, can be displayed as expanded or collapsed.

The following sections define the required UI Automation tree structure, properties, control patterns, and events for the TreeItem control type. The UI Automation requirements apply to all tree item controls, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to tree item controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
TreeItem - CheckBox (0 or 1) - Image (0 or 1) - Button (0 or 1) - TreeItem (0 or more)	TreeItem - TreeItem (0 or more)

Tree item controls can have zero or more tree item children in the content view of the UI Automation tree. If the tree item control has functionality beyond what is exposed in the control patterns listed below, then the control should be based on the Data Item control type.

Collapsed tree items will not display in the control view or content view until they become expanded and visible (or, can be scrolled into view).

The control view can contain additional details for a control, including an associated image or a button. For example, an item in an outline view might contain an image as well as a button to expand or collapse the outline. These detail objects don't appear in the content view because the information is already represented by the parent tree item. Tree items that are scrolled off the screen will appear in both the control and content views of the UI Automation tree and should have the IsOffscreenProperty set to true.

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to list controls. For more information on UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	This property must return a location of the item that will cause the item to change selection state or become focused.
ControlTypeProperty	TreeItem	This value is the same for all UI frameworks.
IsContentElementProperty	True	The list control is always included in the content view of the UI Automation tree.
IsControlElementProperty	True	The list control is always included in the control view of the UI Automation tree.
IsOffscreenProperty	See notes.	This property is set to indicate when a tree item control is scrolled off the screen.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
ItemTypeProperty	See notes.	If the tree item control uses a visual icon to indicate that is a particular type of object, then this property must be supported and indicate what the object is.
LabeledByProperty	`Null`	Tree item controls are self-labeling.
LocalizedControlTypeProperty	"tree item"	Localized string corresponding to the TreeItem control type.
NameProperty	See notes.	This property exposes the text displayed for each tree item control.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by list controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN/PATTERN PROPERTY	SUPPORT/VALUE	NOTES
IInvokeProvider	Depends	Implement this control pattern if the tree item has a separate, actionable command.
IExpandCollapseProvider	Yes	All tree items can be expanded or collapsed.
ExpandCollapseState	Expanded, Collapsed, or Leaf Node	Tree items will be leaf nodes when they are not expanded or collapsed.
IScrollItemProvider	Depends	Implement this control pattern if the tree container supports the Scroll control pattern.
ISelectionItemProvider	Depends	Implement this control pattern if it is possible to have an active selection that is maintained when the user returns to the tree container.
SelectionContainer	Yes	This property will expose the same container for all items within the container.
IToggleProvider	Depends	Implement this control pattern if the tree item has an associated check box.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all tree item controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None
IsEnabledProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
ItemStatusProperty property-changed event.	Depends	None
NameProperty property-changed event.	Required	None
StructureChangedEvent	Required	None

UI AUTOMATION EVENT	SUPPORT	NOTES
ExpandCollapseStateProperty property-changed event.	Required	None
InvokedEvent	Depends	None
CurrentViewProperty property-changed event.	Depends	None
ElementAddedToSelectionEvent	Depends	None
ElementRemovedFromSelectionEvent	Depends	None
ElementSelectedEvent	Depends	None
ToggleStateProperty property-changed event.	Depends	None
ValueProperty property-changed event.	Depends	None

See also

- TreeItem
- UI Automation Control Types Overview
- UI Automation Overview

UI Automation Support for the Window Control Type

1/28/2020 • 2 minutes to read • Edit Online

NOTE

This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

This topic provides information about UI Automation support for the Window control type. In UI Automation, a control type is a set of conditions that a control must meet in order to use the ControlTypeProperty property. The conditions include specific guidelines for UI Automation tree structure, UI Automation property values, and control patterns.

The window control consists of the window frame, which contains child objects such as title bar, client, and other objects.

The UI Automation requirements in the following sections apply to all controls that implement the Window control type, whether Windows Presentation Foundation (WPF), Win32, or Windows Forms.

Required UI Automation Tree Structure

The following table depicts the control view and the content view of the UI Automation tree that pertains to window controls and describes what can be contained in each view. For more information on the UI Automation tree, see UI Automation Tree Overview.

CONTROL VIEW	CONTENT VIEW
Window	Window

Required UI Automation Properties

The following table lists the UI Automation properties whose value or definition is especially relevant to window controls. For more information about UI Automation properties, see UI Automation Properties for Clients.

UI AUTOMATION PROPERTY	VALUE	NOTES
AutomationIdProperty	See notes.	The value of this property needs to be unique across all controls in an application.
BoundingRectangleProperty	See notes.	The outermost rectangle that contains the whole control.
ClickablePointProperty	See notes.	The window control must have a clickable point that will result in causing the window to become selected or unselected.

UI AUTOMATION PROPERTY	VALUE	NOTES
ControlTypeProperty	Window	This value is the same for all UI frameworks.
IsContentElementProperty	True	The window control must always be content.
IsControlElementProperty	True	The window control must always be a control.
IsKeyboardFocusableProperty	See notes.	If the control can receive keyboard focus, it must support this property.
LabeledByProperty	`null`	Window controls do not have a static Window label.
LocalizedControlTypeProperty	"window"	Localized string corresponding to the Window control type.
NameProperty	See notes.	The window control always contains a primary Window element that relates to what the user would associate as the most semantic identifier for the item.

Required UI Automation Control Patterns

The following table lists the UI Automation control patterns required to be supported by window controls. For more information on control patterns, see UI Automation Control Patterns Overview.

CONTROL PATTERN	SUPPORT	NOTES
IDockProvider	Conditional	Must be supported if the window has the ability to be docked.
ITransformProvider	Required	Enables the window to be moved, resized, or rotated on the screen.
IWindowProvider	Required	Enables specific operations for the window.

Required UI Automation Events

The following table lists the UI Automation events required to be supported by all window controls. For more information about events, see UI Automation Events Overview.

UI AUTOMATION EVENT	SUPPORT	NOTES
AsyncContentLoadedEvent	Required	None
AutomationFocusChangedEvent	Required	None
BoundingRectangleProperty property-changed event.	Required	None

UI AUTOMATION EVENT	SUPPORT	NOTES
IsEnabledProperty property-changed event.	Required	None
IsOffscreenProperty property-changed event.	Required	None
LayoutInvalidatedEvent	Required	None
NameProperty property-changed event.	Required	None
StructureChangedEvent	Required	None
HorizontallyScrollableProperty property-changed event.	Depends	None
HorizontalScrollPercentProperty property-changed event.	Depends	None
HorizontalViewSizeProperty property-changed event.	Depends	None
VerticalScrollPercentProperty property-changed event.	Depends	None
VerticallyScrollableProperty property-changed event.	Depends	None
VerticalViewSizeProperty property-changed event.	Depends	None
WindowClosedEvent	Required	None
WindowOpenedEvent	Required	None
WindowVisualStateProperty property-changed event.	Depends	None

See also

- Window
- UI Automation Control Types Overview
- UI Automation Overview