Appendix A

Windows Automation API: Overview

Source: "Windows Automation API SDK" from the Microsoft Developer Network (MSDN) Web site. To view this content online, go to *http://msdn.microsoft.com/en-us/library/aa163327.aspx*.

Windows offers two application programming interface (API) specifications for user interface accessibility and software test automation: Microsoft Active Accessibility, and User Interface Automation (UI Automation). Microsoft Active Accessibility is the legacy API that was introduced in Windows 95 as a platform add-in. UI Automation is a Windows implementation of the User Interface Automation specification.

This section provides a high-level overview of Microsoft Windows Automation API 3.0, which includes the legacy Microsoft Active Accessibility API and the new UI Automation API. The overview highlights the similarities and differences between Microsoft Active Accessibility and UI Automation, describes the components and features that enable the two technologies to work together, and provides guidelines for choosing which technology to implement.

This section includes the following topics:

- Microsoft Active Accessibility and UI Automation Compared
- Architecture and Interoperability
- Limitations of Microsoft Active Accessibility
- UI Automation Specification
- The IAccessibleEX Interface
- Choosing Microsoft Active Accessibility, UI Automation, or IAccessibleEx

Microsoft Active Accessibility and UI Automation Compared

Although Microsoft Active Accessibility and Microsoft UI Automation are two different technologies, the basic design principles are similar. Both expose the UI object model as a hierarchical tree, rooted at the desktop. Microsoft Active Accessibility represents individual UI elements as *accessible objects*, and UI Automation represents them as *automation elements*. Both refer to the accessibility tool or software automation program as the *client*. However, Microsoft Active Accessibility refers to the application or control offering the UI for accessibility as the *server*, while UI Automation refers to this as the *provider*.

Microsoft Active Accessibility offers a single COM interface with a fixed, small set of properties. UI Automation offers a richer set of properties, as well as a set of extended interfaces called Control Patterns to manipulate accessible objects in ways Microsoft Active Accessibility cannot.

While UI Automation previously had both managed and unmanaged APIs for providers, the original release had no unmanaged interfaces for clients. Now, UI Automation clients can be written entirely in unmanaged code.

The latest framework also provides support for transitioning from Microsoft Active Accessibility servers to UI Automation providers. The IAccessibleEx interface specification enables support for specific UI Automation Patterns and Properties to be added to legacy Microsoft Active Accessibility servers without needing to rewrite the entire implementation. The specification also allows in-process Microsoft Active Accessibility clients to access UI Automation provider interfaces directly, rather than through UI Automation client interfaces.

The ecosystem of Windows automation technologies, called the Windows Automation API, includes classic Microsoft Active Accessibility and Windows implementations of the UI Automation specification. The UI Automation specification is implemented on many Microsoft products, including Windows 7, Windows Vista, Windows Server 2008, Windows Presentation Foundation (WPF), and Microsoft Silverlight.

Architecture and Interoperability

This section briefly describes the architecture of the Windows Automation technologies Microsoft Active Accessibility and Microsoft UI Automation, and the components that allow interoperability between applications based on the two different technologies.

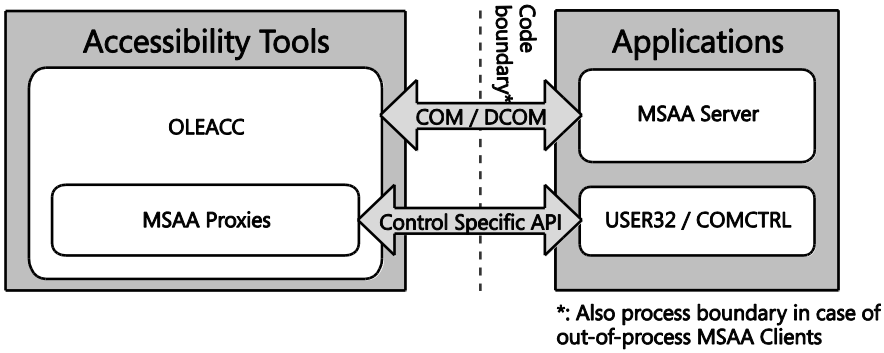
Microsoft Active Accessibility Architecture

Microsoft Active Accessibility exposes basic information about custom controls such as control name, location on screen, and type of control, as well as state information such as visibility and enabled/disabled status. The UI is represented as a hierarchy of accessible objects; changes and actions are represented as WinEvents.

Microsoft Active Accessibility consists of the following components:

- **Accessible object** A logical UI element (such as a button) that is represented by an IAccessible COM interface and an integer child identifier (ChildID).
- **WinEvents** An event system that enables servers to notify clients when an accessible object changes.
- **OLEACC.dll** The run-time, dynamic-link library that provides the Microsoft Active Accessibility API and the accessibility system framework. OLEACC implements proxy objects that provide default accessibility information for standard UI elements, including USER controls, USER menus, and common controls.

For Microsoft Active Accessibility, the system component of the accessibility framework (OLEACC) helps the communication between accessibility tools and applications, as the following illustration shows.



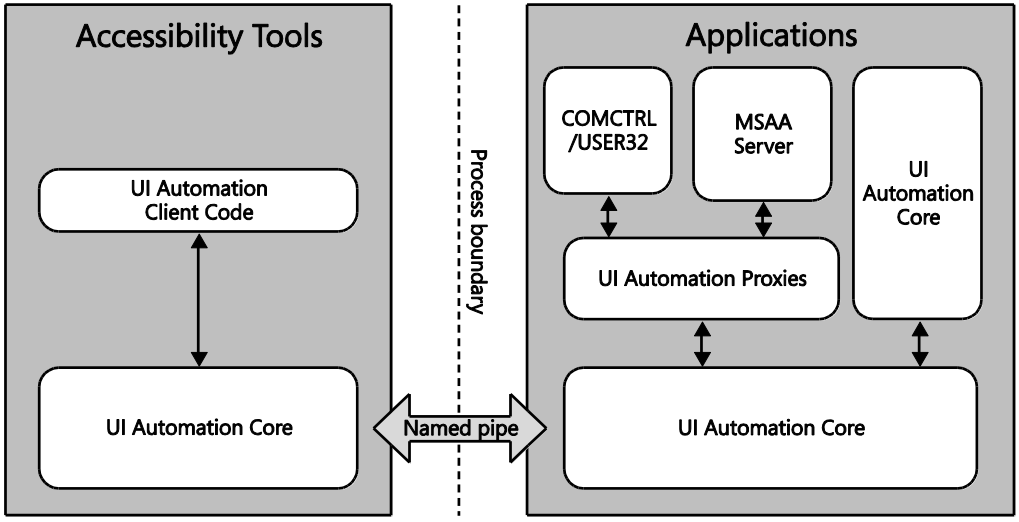
The applications (Microsoft Active Accessibility servers) provide UI accessibility information to tools (Microsoft Active Accessibility clients), which interact with the UI on behalf of users. The code boundary is both a programmatic and a process boundary.

UI Automation Architecture

With UI Automation, the UI Automation Core component (UIAutomationCore.dll) is loaded into both the accessibility tools' and applications' processes. The core component manages cross-process communication, provides higher level services such as searching for elements by Property values, and enables bulk fetching or caching of Properties, which provides better performance than the Microsoft Active Accessibility implementation.

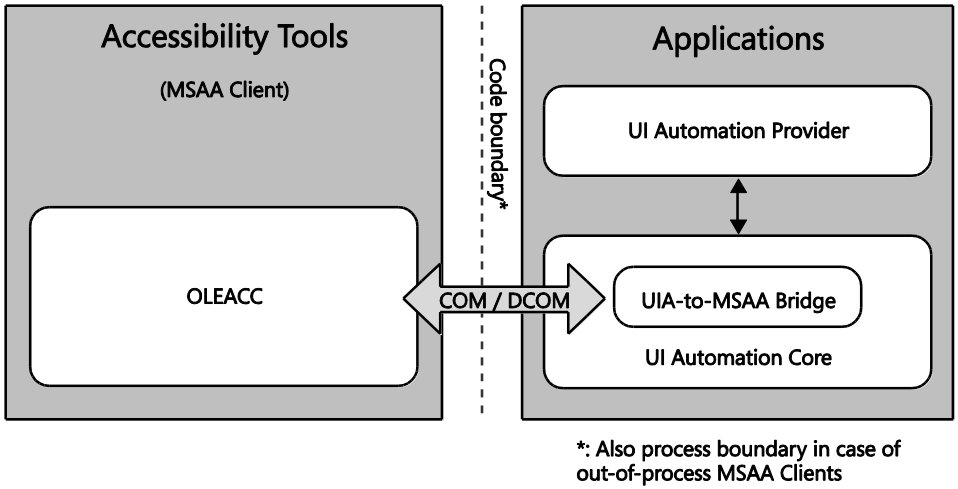
UI Automation includes proxy objects that provide UI information about standard UI elements such as USER controls, USER menus, and common controls. It also includes proxies that enable UI Automation clients to get UI information from Microsoft Active Accessibility servers.

The following illustration shows the relationships among the various components in UI Automation providers (Accessibility Tools) and clients (Applications).

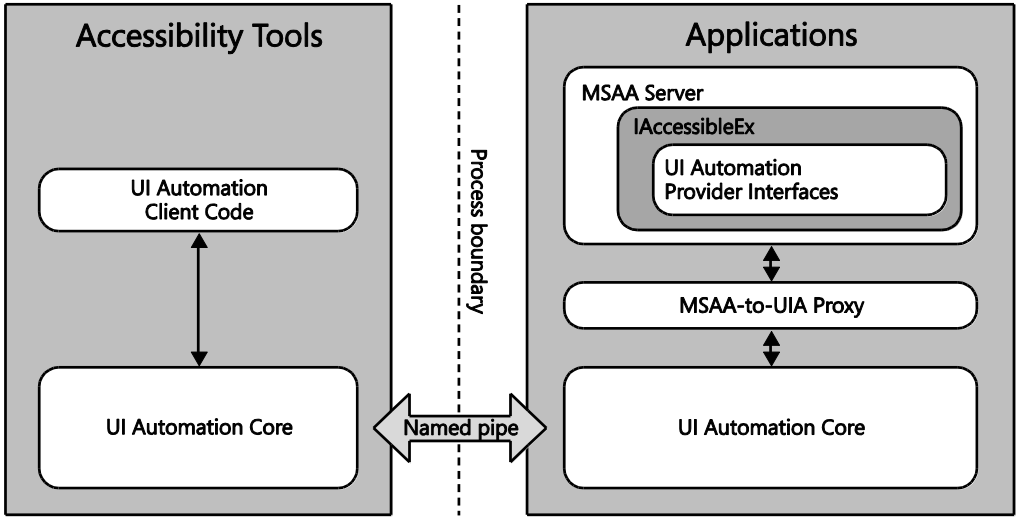


Interoperability Between Microsoft Active Accessibility-Based Applications and UI Automation-Based Applications

The UI Automation to Microsoft Active Accessibility Bridge enables Microsoft Active Accessibility clients to access UI Automation providers by converting the UI Automation object model to a Microsoft Active Accessibility object model. The following illustration shows the role of the UI Automation-to-Microsoft Active Accessibility Bridge.



Similarly, the Microsoft Active Accessibility-to-UI Automation Proxy translates Microsoft Active Accessibility-based server object models for UI Automation clients. The following illustration shows the role of the Microsoft Active Accessibility-to-UI Automation Proxy.



By using the IAccessibleEx interface, you can improve existing Microsoft Active Accessibility Server implementations by adding only required UI Automation object model information. The Microsoft Active Accessibility-to-UI Automation Proxy takes care of incorporating the added UI Automation object model. For more information, see the section of this appendix titled "The IAccessibleEx Interface."

Limitations of Microsoft Active Accessibility

Microsoft designed the Microsoft Active Accessibility object model about the same time as Windows 95 released. The model is based on "roles" defined a decade ago, and you cannot support new UI behaviors or merge two or more roles together. There is no text object model, for example, to help assistive technologies deal with complex Web content. UI Automation overcomes these limitations by introducing Control Patterns that enable objects to support more than one role, and the UI Automation Text Control Pattern offers a full-fledged text object model.

Another limitation involves navigating the object model. Microsoft Active Accessibility represents the UI as a hierarchy of accessible objects. Clients navigate from one accessible object to another using interfaces and methods available from the accessible object. Servers can expose the children of an accessible object with properties of the IAccessible interface, or with the standard IEnumVARIANT COM interface. Clients, however, must be able to deal with both approaches for any server. This ambiguity means extra work for client implementers, and broken accessible object models for server implementers.

UI Automation represents the UI as a hierarchical tree of Automation Elements, and provides a single interface for navigating the tree. Clients can customize the view of elements in the tree by scoping and filtering.

Finally, Microsoft Active Accessibility properties and functions cannot be extended without breaking or changing the IAccessible COM interface specification. The result is that new control behavior cannot be exposed through the object model; it tends to be static.

With UI Automation, as new UI elements are created, application developers can introduce custom Properties, Control Patterns, and Events to describe the new elements.

UI Automation Specification

The UI Automation specification provides flexible programmatic access to UI elements on the Windows 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. The specification can be supported across platforms other than Windows.

The implementation of UI Automation specification in Windows is also called UI Automation (UI Automation). UI Automation is broader in scope than just an interface definition. UI Automation provides:

- An object model and functions that make it easy for client applications to receive events, retrieve property values, and manipulate UI elements.
- A core infrastructure for finding and fetching across process boundaries.
- A set of interfaces for providers to express the tree structure, general properties, and functionality of UI elements.
- A "Control Type" property that allows clients and providers to clearly indicate the common properties, functionality, and structure of a UI object.

UI Automation improves on Microsoft Active Accessibility by:

- Enabling efficient out-of-process clients, while continuing to allow in-process access.
- Exposing more information about the UI in a way that allows clients to be out-of-process.
- Coexisting with and leveraging Microsoft Active Accessibility without inheriting its limitations. For more information, see the section of this appendix titled "Limitations of Microsoft Active Accessibility."

The implementation of the UI Automation specification in Windows features COM-based interfaces and managed interfaces.

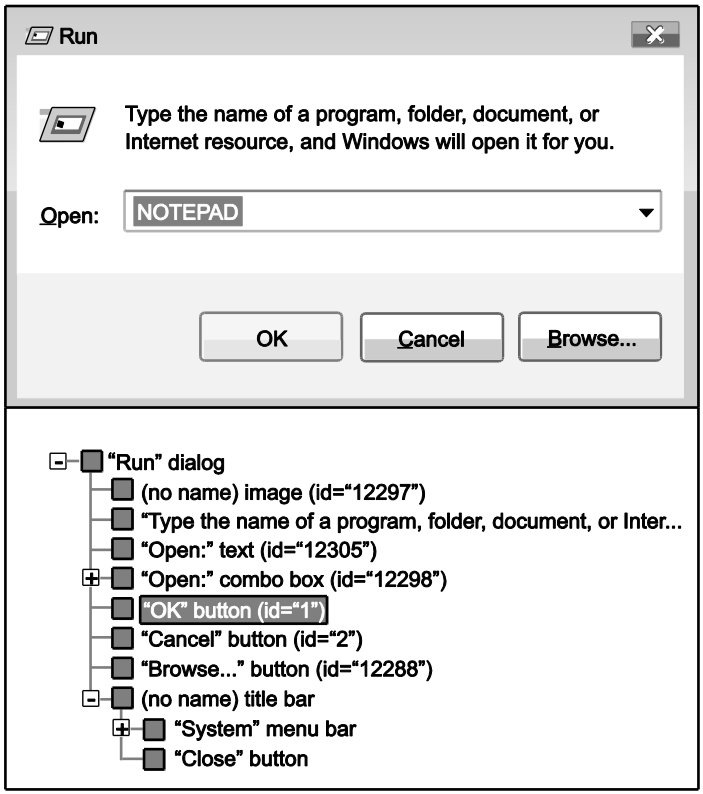
UI Automation Elements

UI Automation exposes every piece of the UI to client applications as an *automation element*. Providers supply Property values for each element. Elements are exposed as a tree structure, with the desktop as the root element.

Automation Elements expose common properties of the UI elements they represent. One of these properties is the Control Type, which describes its basic appearance and functionality (for example, a button or a check box).

UI Automation Tree

The UI Automation tree represents the entire UI: the root element is the current desktop, and child elements are application windows. Each of these child elements can contain elements representing menus, buttons, toolbars, and so on. These elements in turn can contain elements like list items, as the following illustration shows.



Be aware that the order of the siblings in the UI Automation tree is quite important. Objects that are next to each other visually should also be next to each other in the UI Automation tree.

UI Automation providers for a particular control support navigation among the child elements of that control. However, providers are not concerned with navigation between these control sub-trees. This is managed by the UI Automation core, using information from the default window providers.

To help clients process UI information more effectively, the framework supports alternative views of the automation tree: raw view, control view, and content view. As the following table shows, the type of filtering determines the views, and the client defines the scope of a view.

Automation Tree	Description
Raw view	The full tree of Automation Element objects for which the desktop is the root.
Control view	A subset of the raw view that closely maps to the UI structure as the user perceives it.
Content view	A subset of the control view that contains content most relevant to the user, like the values in a drop-down combo box.

UI Automation Properties

The UI Automation specification defines two kinds of properties: Automation Element Properties and Control Pattern Properties. Automation Element Properties apply to most controls, providing fundamental information about the element, such as its name. Control Pattern Properties apply to Control Patterns, which are described next.

Unlike with Microsoft Active Accessibility, every UI Automation Property is identified by a GUID and a programmatic name, which makes new Properties easier to introduce.

UI Automation Control Patterns

A Control Pattern describes a particular aspect of the functionality of an Automation Element. For example, a simple "click-able" control like a button or hyperlink should support the Invoke Control Pattern to represent the "click" action.

Each Control Pattern is a canonical representation of possible UI features and functions. The current implementation of UI Automation defines 22 Control Patterns. The Windows Automation API can also support custom Control Patterns. Unlike Microsoft Active Accessibility role or state properties, one Automation Element can support multiple UI Automation Control Patterns.

UI Automation Control Types

A Control Type is an Automation Element Property that specifies a well-known control that the element represents. Currently, UI Automation defines 38 Control Types, including Button, CheckBox, ComboBox, DataGrid, Document, Hyperlink, Image, ToolTip, Tree, and Window.

Before you can assign a Control Type to an element, the element needs to meet certain conditions, including a particular automation tree structure, Property values, Control Patterns, and Events. However, you are not limited to these. You can extend a control with custom Patterns and Properties, as well as with the pre-defined ones.

The total number of pre-defined Control Types is significantly lower than Microsoft Active Accessibility accRole definitions, because UI Automation Control Types can be combined to express a larger set of features while Microsoft Active Accessibility roles cannot.

UI Automation Events

UI Automation Events notify applications of changes to, and actions taken with Automation Elements. There are four different types of UI Automation Events, and they do not necessarily mean that the visual state of the UI has changed. The UI Automation Event model is independent of the WinEvent framework in Windows, although the Windows Automation API makes UI Automation Events interoperable with the Microsoft Active Accessibility framework.

The IAccessibleEx Interface

The IAccessibleEx interface enables existing applications or UI libraries to extend their Microsoft Active Accessibility object model to support UI Automation without rewriting the implementation from scratch. With IAccessibleEx, you can implement only the additional UI Automation Properties and Control Patterns needed to fully describe the UI and its functionality.

Because the Microsoft Active Accessibility-to-UI Automation Proxy translates the object models of IAccessibleEx-enabled Microsoft Active Accessibility servers as UI Automation object models, UI Automation clients do not need to do any extra work. The IAccessibleEx interface can also enable in-process Microsoft Active Accessibility clients to interact directly with UI Automation providers.

Choosing Microsoft Active Accessibility, UI Automation, or IAccessibleEx

If you are developing a new application or control, Microsoft recommends using UI Automation. Although Microsoft Active Accessibility can be easier to implement in the short term, the limitations inherent in this technology, such as its aging object model and inability to support new UI behaviors or merge rolls, makes it more difficult and costly over the long term. These limitations become especially apparent when introducing new controls. For more information, see the section of this appendix titled “Limitations of Microsoft Active Accessibility.”

The UI Automation object model is easier to use and is more flexible than that of Microsoft Active Accessibility. The UI Automation Elements reflect the evolution of modern user interfaces, and developers can define custom UI Automation Control Patterns, Properties, and Events.

Microsoft Active Accessibility tends to run slowly for clients that run out of process. To improve performance, developers of accessibility tool programs often choose to hook into and run their programs in the target application process: an extremely difficult and risky approach. UI Automation is much easier to implement for out-of-process clients, and offers much better performance and reliability.

If you are updating an existing Microsoft Active Accessibility-based application or control, consider adding support for UI Automation by implementing the IAccessibleEx interface. First, ensure that your application or control meets the following requirements:

- The baseline Microsoft Active Accessibility server's hierarchy of accessible objects must be well-organized and error-free. IAccessibleEx cannot fix problems with existing accessible object hierarchies.
- Your IAccessibleEx implementation must comply with both the Microsoft Active Accessibility specification, and the UI Automation specification. Microsoft provides a set of tools for validating compliance with both specifications.

If either of these requirements is not met, consider implementing UI Automation natively. You can keep legacy Microsoft Active Accessibility server implementations for backward compatibility if it is necessary. From a UI Automation client's perspective, there is no difference between UI Automation providers and Microsoft Active Accessibility servers that implement IAccessibleEx correctly.

Appendix B

UI Automation Overview

Source: "Windows Automation API SDK" from the Microsoft Developer Network (MSDN) Web site. To view this content online, go to *http://msdn.microsoft.com/en-us/library/aa163327.aspx*.

Microsoft UI Automation is an accessibility framework for Windows. It provides programmatic access to most user interface (UI) elements on the desktop. It enables 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.

UI Automation was first available in Windows XP as part of the Microsoft .NET Framework. Although an unmanaged C++ API was also published at that time, the usefulness of client functions was limited because of interoperability issues. For Windows 7, the API has been rewritten in the Component Object Model (COM).

> **Note** Although the library functions introduced in the earlier version of UI Automation are still documented, they should not be used in new applications.

UI Automation client applications can be written with the assurance that they will work on multiple Windows control frameworks. The UI Automation core masks any differences in the frameworks that underlie various pieces of the UI. For example, the `Content` property of a Windows Presentation Foundation (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 in Windows XP, Windows Server 2003, and later operating systems.

UI Automation providers are components that implement UI Automation support on controls and offer some support for Microsoft Active Accessibility client applications, through a built-in bridging service.

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

This appendix contains the following sections:

- UI Automation Components
- UI Automation Header Files
- UI Automation Model
- UI Automation Providers

UI Automation Components

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

Component	Description
Provider API	A set of COM interfaces that are implemented by UI Automation providers. UI Automation providers are objects that provide information about UI elements and respond to programmatic input.
Client API	A set of COM interfaces that enable client applications to obtain information about the UI and to send input to controls. **Note** The functions described in Deprecated Control Pattern Functions and Deprecated Node Functions are obsolete and in the process of being removed. Instead, client applications should use the UI Automation COM interfaces described in UI Automation Element Interfaces for Clients.
UiAutomationCore.dll	The run-time library, sometimes called the UI Automation core, that handles communication between providers and clients.
OLEACC.dll	The run-time library for Microsoft Active Accessibility and the proxy objects. The library also provides proxy objects used by the MSAA-to-UIA Proxy to support Win32 controls.

There are two ways of using UI Automation: to create support for custom controls by using the provider API, and to create client applications that use the UI Automation core to communicate with UI elements. Depending on your focus, you should refer to different parts of the documentation.

UI Automation Header Files

The UI Automation API is defined in several different C/C++ header files that are included with the Microsoft Windows Software Development Kit (SDK). The UI Automation header files are described in the following table.

Header file	Description
uiautomationclient.h	Defines the interfaces and related programming elements used by UI Automation clients.
uiautomationcore.h	Defines the interfaces and related programming elements used by UI Automation providers.
uiautomationcoreapi.h	Defines general constants, GUIDs, data types, and structures used by UI Automation clients and providers. It also contains definitions for the deprecated node and Control Pattern functions.
uiautomation.h	Includes all of the other UI Automation header files. Because most UI Automation applications require elements from all UI Automation header files, it is best to include uiautomation.h in your UI Automation application projects instead of including each file individually.

If you are developing an application that uses the UI Automation API, you should include uiautomation.h in your project. If your application supports Microsoft Active Accessibility, include the oleacc.h header file. UI Automation applications that use GUIDs also require the initguid.h header file. If needed, initguid.h should be included before uiautomation.h.

UI Automation Model

UI Automation exposes every element of the UI to client applications as an object represented by the IUIAutomationElement interface. 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. These standard views of the structure can easily be seen by using the UI Spy application that is included with the Windows SDK. Applications can also create custom views.

A UI Automation Element exposes properties of the control or UI element that it represents. One of these properties is the Control Type, which defines the basic appearance and functionality of the control or UI element as a single recognizable entity, for example, a button or check box.

In addition, a UI Automation Element exposes one or more Control Patterns. A Control Pattern provides a set of Properties that are specific to a particular Control Type. A Control Pattern also exposes methods that enable client applications to get more information about the element and to provide input to the element.

> **Note** There is no 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. However, a control can exhibit only a single Control Type.

UI Automation 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 Properties and Control Pattern information be passed to 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.

Go further: Go to http://go.microsoft.com/fwlink/?LinkId=150842 for more information on the following topics:

- Deprecated Control Pattern Functions
- Deprecated Node Functions
- UI Automation Element Interfaces for Clients
- UI Automation Control Types Overview
- UI Automation Control Patterns Overview
- UI Automation Events Overview

UI Automation Providers

After designing your implementation, you must implement a provider interface to support your implementation. For more details on how to do so, go to *http://go.microsoft.com/fwlink/?LinkId=150842*.