PD ISO/IEC TR 13066-2:2016

BSI Standards Publication

Information technology — Interoperability with assistive technology (AT)

Part 2: Windows accessibility application programming interface (API)

bsi.

...making excellence a habit.™

Document Sharing Hub

Full document details: PD ISO/IEC TR 13066-2:2016

National foreword

This Published Document is the UK implementation of ISO/IEC TR 13066-2:2016. It supersedes PD ISO/IEC TR 13066-2:2012 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee ICT/-/6, ICT Accessibility.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2016.
Published by BSI Standards Limited 2016

ISBN 978 0 580 87486 4
ICS 11.180.99; 35.180

Compliance with a British Standard cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 29 February 2016.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 13066-2:2016

TECHNICAL REPORT

ISO/IEC TR 13066-2

Second edition
2016-02-01

Information technology — Interoperability with assistive technology (AT) —

Part 2: Windows accessibility application programming interface (API)

Technologies de l'information — Interopérabilité avec les technologies d'assistance —

Partie 2: Interface de programmation d'applications (API) d'accessibilité Windows

Reference number
ISO/IEC TR 13066-2:2016(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2016

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2016, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 13066-2:2016

Contents

Page

Foreword ... **v**

Introduction ... **vi**

1 Scope ... **1**

2 Terms and definitions ... **1**

3 General description and architecture of the Microsoft Windows Automation API ... **7**
3.1 General description ... 7
3.1.1 Microsoft Active Accessibility overview ... 7
3.1.2 UI Automation overview ... 8
3.1.3 IAccessibleEx interface ... 9
3.2 Architecture ... 10

4 Using the API ... **12**
4.1 Using the Microsoft Active Accessibility API ... 12
4.1.1 Types of Microsoft Active Accessibility support ... 13
4.1.2 Retrieving an accessible object ... 13
4.1.3 The WM_GETOBJECT message ... 13
4.1.4 Special values of Object Identifier ... 14
4.2 Using the UI Automation API ... 15
4.2.1 UI Automation model ... 15
4.2.2 UI Automation tree ... 16
4.2.3 UI Automation control patterns, control types, properties, and events ... 16
4.3 Using the IAccessibleEx interface ... 21
4.3.1 The IAccessibleEx interface implementation ... 21

5 Exposing User Interface Element Information ... **24**
5.1 General ... 24
5.2 Exposing UI Elements with Microsoft Active Accessibility ... 25
5.2.1 How an MSAA Server exposes relevant properties ... 25
5.2.2 Provide support for the Accessible Object structure ... 26
5.2.3 Support hit testing ... 27
5.2.4 Generate appropriate WinEvents ... 27
5.2.5 Object identifier ... 27
5.2.6 How MSAA clients access exposed UI elements ... 28
5.3 Exposing UI Elements with UI Automation ... 28
5.3.1 Types of providers ... 28
5.3.2 UI Automation provider concepts ... 28
5.3.3 Provider interfaces ... 29
5.3.4 Property values ... 30
5.3.5 Provider navigation ... 30
5.3.6 Provider reparenting ... 31
5.3.7 Provider repositioning ... 31
5.3.8 How UI Automation clients access exposed UI Elements ... 32

6 Exposing UI Element actions ... **33**
6.1 Exposing UI Element actions in MSAA ... 33
6.2 Exposing UI Element actions in UI Automation ... 33
6.2.1 UI Automation control pattern components ... 33
6.2.2 Control patterns in providers and clients ... 34
6.2.3 Dynamic control patterns ... 34
6.2.4 Control patterns and related interfaces ... 34

7 Keyboard focus ... **36**
7.1 MSAA keyboard focus and selection ... 36
7.1.1 Focus and selection properties and methods ... 36
7.1.2 Events triggered in menus ... 37
7.2 UI Automation keyboard focus and selection ... 37

Documents
Document Sharing Hub

© ISO/IEC 2016 – All rights reserved

Full document details: PD ISO/IEC TR 13066-2:2016

7.2.1 Focus 37
7.2.2 Selection 38

8 Events 44
8.1 WinEvents 44
8.1.1 USER's role in WinEvents 44
8.1.2 Receiving event notifications 45
8.1.3 Sending events 45
8.1.4 The allocation of WinEvent IDs 45
8.2 UI Automation events 46
8.2.1 How providers raise events 47
8.2.2 How clients register for and process events 48

9 Programmatic modifications of states, properties, values, and text 48
9.1 UI Automation specifications 48
9.1.1 Introduction 48
9.1.2 UI Automation elements 49
9.1.3 UI Automation tree 49
9.1.4 UI Automation properties 50
9.1.5 UI Automation control patterns 50
9.1.6 UI Automation control types 50
9.1.7 UI Automation events 50

10 Design considerations 51
10.1 UI Automation design considerations 51
10.1.1 UI Automation clients 51
10.1.2 UI Automation providers 54
10.1.3 Coexistence and interoperability with Microsoft Active Accessibility 57
10.2 IAccessibleEx design considerations 58
10.2.1 Design consideration for providers before implementing the IAccessibleEx interface 58
10.2.2 IAccessibleEx interface for providers 58
10.2.3 IAccessibleEx interface for clients 59

11 Further Information 63
11.1 Microsoft Active Accessibility and Extensibility 63
11.2 UI Automation extensibility features 63
11.2.1 Registration of custom UI Automation properties, events, and control patterns 63
11.2.2 How clients and providers support custom control patterns 64

Annex A (informative) **Microsoft Active Accessibility to Automation Proxy 65**

Annex B (informative) **UI Automation to Microsoft Active Accessibility Bridge 72**

Annex C (informative) **UI Automation for W3C Accessible Rich Internet Applications (ARIA) Specification 77**

Annex D (informative) **Other Useful APIs for Development and Support of Assistive Technologies 81**

Bibliography 88

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: Foreword — Supplementary information.

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 35, *User interfaces*.

This second edition cancels and replaces the first edition (ISO/IEC/TR 13066–2:2012), which has been technically revised.

ISO/IEC/TR 13066 consists of the following parts, under the general title *Information technology — Interoperability with assistive technology (AT)*:

— *Part 1: Requirements and recommendations for interoperability*

— *Part 2: Windows accessibility application programming interface (API)*

— *Part 3: IAccessible2 accessibility application programming interface (API)*

— *Part 4: Linux/UNIX graphical environments accessibility API*

— *Part 6: Java accessibility application programming interface (API)*

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

Introduction

Individuals with a wide range of functional disabilities, impairments, and difficulties require specific technology to enable computers and software to be accessible to them. This part of ISO/IEC 13066 provides information about the Microsoft® Windows® Automation Frameworks, including Microsoft Active Accessibility, User Interface (UI) Automation, and the common interfaces of these accessibility frameworks including the `IAccessibleEx` interface specification.

The intent of this part of ISO/IEC 13066 is to provide information and application programming interfaces (APIs) needed to use these frameworks. A primary goal of this part of ISO/IEC 13066 is to ensure that accessible software applications can be written in such a way that they are fully compatible with the Microsoft Accessibility APIs available on the Microsoft Windows operating system.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

TECHNICAL REPORT ISO/IEC TR 13066-2:2016(E)

Information technology — Interoperability with assistive technology (AT) —

Part 2: Windows accessibility application programming interface (API)

1 Scope

This part of ISO/IEC 13066 specifies services provided in the Microsoft Windows platform to enable assistive technologies (AT) to interact with other software. One goal of this part of ISO/IEC 13066 is to define a set of application programming interfaces (APIs) for allowing software applications to enable accessible technologies on the Microsoft Windows platform. Another goal of this part of ISO/IEC 13066 is to facilitate extensibility and interoperability by enabling implementations by multiple vendors on multiple platforms.

This part of ISO/IEC 13066 is applicable to the broad range of ergonomics and how ergonomics apply to human interaction with software systems.

2 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

2.1
application programming interface
API
standard set of documented and supported routines that expose operating system programming interfaces and services to applications

Note 1 to entry: An API is usually a source code interface that an operating system, library, or service provides to support requests made by computer programs.

EXAMPLE Examples of operating system services that are exposed by APIs include administration and management, diagnostics, graphics and multimedia, networking, security, system services, user interfaces, and accessibility.

2.2
accessibility
degree to which a computer system is easy to use by all people, including those with disabilities

2.3
accessible object
part of user interface object that is accessible by Microsoft Active Accessibility

Note 1 to entry: An accessible object is represented by an `IAccessible` interface and a `ChildId` identifier

2.4
Accessible Rich Internet Applications
ARIA
accessibility framework from W3C that exposes web content to assistive technologies such as screen readers and speech commanding programs

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

2.5
Assistive Technology
AT
technology designed to provide accessibility support to individuals with physical or cognitive impairments or disabilities

Note 1 to entry: Assistive Technology can be manifested through both hardware and software.

2.6
Accessibility Toolkit
Linux Accessibility Toolkit
ATK
programming support accessibility features in their applications

2.7
automation
replacement of manual operations by computerized methods

Note 1 to entry: With respect to this part of ISO/IEC 13066, automation is a way to manipulate an application's user interface from outside the application.

2.8
automation element
object or element that is accessible by the UI Automation object model

Note 1 to entry: Similar to accessible objects in Microsoft Active Accessibility, an automation element in UI Automation represents a piece or a part of the user interface, such as button, window, or desktop.

2.9
Audio Video Interleaved
AVI
format that enables both audio and video data in a file container

2.10
C#
programming language designed for building applications that run on the .NET Framework

Note 1 to entry: C#, which is an evolution of C and C++, is type safe and object-oriented.

Note 2 to entry: Because it is compiled as managed code, it benefits from the services of the common language runtime, such as language interoperability, security, and garbage collection.

2.11
callback function
function or procedure that third party or client code supplies to a component, often by passing a function pointer through the component's API

Note 1 to entry: The component may then call this code at specific times. This technique is often used by components to signal client code that some event has taken place, or to request client code to perform some specific task.

2.12
clients
component that uses the services of another component

Note 1 to entry: In this part of ISO/IEC 13066, client refers more specifically to a component that uses the services of Microsoft Active Accessibility or UI Automation, or both, to access, identify, or manipulate the user interface (UI) elements of an application

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

2.13
Common Language Runtime
CLR
Microsoft's commercial implementation of the Common Language Infrastructure (CLI) specification

Note 1 to entry: The CLI provides a specification for executable code and the execution environment in which it runs

Note 2 to entry: At the centre of the CLI are a unified type system, a virtual execution system, and a specification for multiple programming languages to share the type system and compile into a common intermediate language.

2.14
Component Object Model
COM
object-oriented programming model that defines how objects interact within a single process or between processes

Note 1 to entry: In COM, clients have access to an object through interfaces implemented on the object.

2.15
content view
subset of the control view of the UI Automation tree

Note 1 to entry: The content view contains UI items that convey the actual 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.

2.16
control pattern
<UI Automation> design implementation that describes a discrete piece of functionality for a control

Note 1 to entry: This functionality can include the visual appearance of a control and the actions it can perform.

2.17
control view
subset of the raw view of the UI Automation tree

Note 1 to entry: The control view includes the UI items that provide information to the user or enable the user to perform an action.

2.18
enumerator
object that iterates through its associated collection

Note 1 to entry: An enumerator can be thought of as a movable pointer to any element in the collection.

2.19
Global Unique Identifier
GUID
unique reference number used as an identifier in computer software

2.20
HWND
unique long integer value that is assigned by Microsoft Windows to the current window, where a window is a primitive of Windows' UI management

2.21
in-process
<UI Automation> Microsoft Accessibility code that is executed in a target application's process

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

2.22
Java Accessibility Application Programming Interface
JAAPI
accessibility framework for the Java™ SE platform that exposes Java applications to assistive technologies such as screen readers and speech commanding programs

2.23
Java Development Kit
JDK
collection of programming tools

2.24
Java Virtual Machine
JVM
environment in which Java bytecode can be executed

2.25
managed API
API that, when compiled and run, is under the control of an intermediary runtime infrastructure, like a virtual machine

EXAMPLE Microsoft's Common Language Runtime (CLR) and the Java Virtual Machine (JVM) are examples of runtime infrastructures.

Note 1 to entry: Managed code is compiled into an intermediate language construct (for example, byte code) and the runtime infrastructure handles the compilation into machine code. The runtime infrastructure handles programming constructs like memory management.

2.26
Microsoft Active Accessibility
COM-based technology that improves the way accessibility aids work with applications running on Microsoft Windows

Note 1 to entry: Microsoft Active Accessibility provides dynamic-link libraries (DLLs) that are incorporated into the operating system, as well as a COM interface and application programming elements that provide reliable methods for exposing information about user interface elements.

2.27
MSDN
the Microsoft Developer Network, which is a technical information resource for developers who are using Microsoft technologies

2.28
Multiple Document Interface
MDI
A document interface that allows a window to reside under a parent window

2.29
native API
API that, when compiled and run, is not under the control of an intermediary runtime infrastructure such as a virtual machine or CLR

Note 1 to entry: Native code compiles directly to machine code, and the developer is responsible for most aspects of programming constructs (for example, pointers, freeing memory, and so on). Also known as a native API.

2.30
out-of-process
<UI Automation> Microsoft Accessibility code that is executed in a different process from the target application's process

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

2.31
providers
<UI Automation> components that expose information about UI elements

EXAMPLE Such components can be applications, DLLs, and so on.

Note 1 to entry: These include any control, module, or application which implements UI Automation provider interfaces.

2.32
raw view
full tree of UI Automation element objects in the UI Automation tree for which the desktop is the root

Note 1 to entry: The raw view closely follows the native programmatic structure of an application and, therefore, is the most accurate view of the UI structure. It is also the base on which the other views of the tree are built.

2.33
root element
element of the UI Automation tree that represents the current desktop and whose child elements represent application windows

Note 1 to entry: Each of these child elements can contain elements representing pieces of UI such as menus, buttons, toolbars, and list boxes.

2.34
servers
components of Microsoft Active Accessibility that have UI elements and expose information about the UI elements and/or allow them to be manipulated

EXAMPLE Such components can be applications, DLLs, and so on.

Note 1 to entry: A Microsoft Active Accessibility server has the same role as a UI Automation provider.

2.35
simple element
<Microsoft Active Accessibility> element that shares an `IAccessible` object with other peer elements

Note 1 to entry: A simple element relies on the shared `IAccessible` object (typically its parent in the object hierarchy) to expose its properties.

2.36
Services Control Manager
SCM
system process that maintains a database of installed services and driver services, and provides a unified and secure means of controlling them

Note 1 to entry: The database includes information on how each service or driver service should be started. It also enables system administrators to customize security requirements for each service and thereby control access to the service.

2.37
system service
application conforming to the interface rules of the Service Control Manager (SCM)

Note 1 to entry: It can be started automatically at system boot, by a user through the Services control panel applet, or by an application that uses the service functions.

Note 2 to entry: Services can execute even when no user is logged on to the system. File services, indexing service, memory management, power management, and remote desktop services are examples of services.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

2.38
Text Services Framework
TSF
simple and scalable framework for the delivery of advanced text input and natural language technologies

Note 1 to entry: TSF can be enabled in applications, or as a TSF text service.

Note 2 to entry: A TSF text service provides multilingual support and delivers text services such as keyboard processors, handwriting recognition, and speech recognition.

2.39
user interface
UI
mechanisms by which a person interacts with a computer system

Note 1 to entry: The user interface provides input mechanisms, allowing users to manipulate a system.

Note 2 to entry: It also provides output mechanisms, allowing the system to produce the effects of the users' manipulation.

2.40
User Interface Automation
UI Automation
UIA
accessibility framework that exposes applications to software automation or to assistive technologies such as screen readers and speech commanding programs

2.41
virtual machine
VM
computer within a computer, implemented in software

Note 1 to entry: A virtual machine emulates a complete hardware system, from processor to network card, in a self-contained, isolated software environment, enabling the simultaneous operation of otherwise incompatible operating systems

Note 2 to entry: Each operating system runs in its own isolated software partition.

2.42
Visual Basic
VB
visual programming environment from Microsoft based on the BASIC programming language

2.43
Web Accessibility Initiative
WAI
effort to improve the accessibility of the World Wide Web

2.44
WinEvents
mechanism that allows servers and the Windows operating system to notify clients when an accessible object changes

2.45
World Wide Web Consortium
W3C
A standards organization for the World Wide Web

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

3 General description and architecture of the Microsoft Windows Automation API

3.1 General description

The Microsoft® Windows® Automation API consists of two accessibility frameworks, Microsoft Active Accessibility and User Interface Automation (UI Automation). The `IAccessibleEx` interface specification integrates the two accessibility frameworks.

Although Microsoft Active Accessibility and UI Automation are two different frameworks, the basic design principles are similar. The purpose of both is to expose rich information about the UI elements used in Windows applications. Developers of accessibility tools can use this information to help make applications more accessible to people with vision, hearing, or motion disabilities.

3.1.1 Microsoft Active Accessibility overview

Microsoft Active Accessibility is based on the Component Object Model (COM), which defines a common way for applications and operating systems to communicate. The goal of Microsoft Active Accessibility is to allow custom controls to expose basic information, such as name, location on screen, or type of control, and state information such as visibility, enabled, or selected.

The *accessible object* is the central object of Microsoft Active Accessibility and is represented by an `IAccessible` COM interface and an integer `ChildId`. It allows applications to expose UI elements as a tree structure that represents the structure of the UI. Each element of this tree exposes a set of properties and methods that allow the corresponding UI element to be manipulated. Microsoft Active Accessibility clients can access the programmatic UI information through a standard API. The following sections describe the main parts of Microsoft Active Accessibility, including accessible objects, the WinEvents mechanism, the Microsoft Active Accessibility runtime (`Oleacc.dll`), and Microsoft Active Accessibility clients and servers.

3.1.1.1 Microsoft Active Accessibility components

Microsoft Active Accessibility contains the following main components.

— Accessible Object — A logical UI element (such as a button) that is represented by an `IAccessible` COM interface and a `ChildId` value.

  — The `IAccessible` interface has properties and methods for obtaining information about and manipulating UI elements.

  — `ChildId` is an identifier for an accessible object that is used together with an `IAccessible` instance to refer to a specific UI element.

— *WinEvents* — An event system that allows servers to notify clients when an accessible object changes. For more information, see Clause 8.

— `Oleacc.dll` — A run-time dynamic-link library that provides the Microsoft Active Accessibility API and the accessibility system framework. `Oleacc.dll` also provides proxy objects for the Windows operating system standard controls.

3.1.1.2 Oleacc.dll

The following APIs and functions are included in `Oleacc.dll`.

— Client APIs — APIs that clients use to request an `IAccessible` interface pointer (for example, `AccessibleObjectFromX`).

— Server APIs — APIs that servers use to return an `IAccessible` interface pointer to a client (for example, `LresultFromObject`).

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

— APIs for getting localized text for the role and state codes (for example, GetRoleText and `GetStateText`).

— Helper APIs (for example, `AccessibleChildren`).

— Proxies — Code that provides the default implementation of an `IAccessible` interface for standard USER and COMCTL controls. Because these controls implement the `IAccessible` interface on behalf of the system controls, they are known as *proxies*.

3.1.1.3 Microsoft Active Accessibility clients

Microsoft Active Accessibility helps accessibility aids, called *clients*, interact with standard and custom UI elements of other applications and the operating system. Clients can use Microsoft Active Accessibility to access, identify, and manipulate an application's UI elements. Clients include accessibility aids, automated testing tools, and computer-based training applications.

Clients must know when the server's UI changes so that information can be conveyed to the user. They are notified about changes in the server UI by registering to receive notifications of specific changes through a mechanism called *Window Events*, or *WinEvents*. For more information, see Clause 8.

To learn about and manipulate a particular UI element, clients use a pair consisting of an `IAccessible` interface and a `ChildId`.

3.1.1.4 Microsoft Active Accessibility servers

Applications that interact with and provide information to clients are called *servers.* Servers include any control, module, or application that implements Microsoft Active Accessibility. A server uses Microsoft Active Accessibility to provide information about its UI elements to clients.

3.1.2 UI Automation overview

UI Automation provides programmatic access to 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. The UI Automation Specification is designed so that it can be supported across platforms other than Microsoft Windows.

UI Automation is broader in scope than just an interface definition. UI Automation provides

— a set of classes that make it easy for client applications to receive events, retrieve property values, and manipulate UI elements,

— a core infrastructure for doing fetch, find, and similar operations efficiently across process boundaries,

— a set of interfaces for providers to express the UI as a tree structure, along with some general properties,

— a set of interfaces that providers use to express other properties and functionality specific to the control type. These are the control pattern interfaces.

To improve on Microsoft Active Accessibility, UI Automation aims to address the following goals:

— enable efficient *out-of-process* clients, while continuing to allow *in-process* access;

— expose more information about the UI in a way that allows clients to be out-of-process;

— coexist with and use Microsoft Active Accessibility, but do not inherit problems that exist in Microsoft Active Accessibility;

— provide an alternative to `IAccessible` that is simple to implement.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

The Microsoft Windows implementation of UI Automation features COM-based interfaces and managed interfaces that are included with the Microsoft .NET Framework.

3.1.2.1 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.
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 Microsoft Active Accessibility to UI Automation Proxy to support Win32 controls.

UI Automation can be used 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.

3.1.2.2 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. 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.

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.

3.1.3 IAccessibleEx interface

Controls that do not have a Microsoft UI Automation provider, but that implement the Microsoft Active Accessibility `IAccessible` interface, can easily be upgraded to provide some UI Automation functionality by implementing the `IAccessibleEx` interface. This interface enables the control to expose UI Automation properties and control patterns, without the need for a full implementation of UI Automation provider interfaces such as `IRawElementProviderFragment`.

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`, developers can implement only the additional UI Automation properties and control patterns needed to fully describe the UI and its functionality.

© ISO/IEC 2016 – All rights reserved

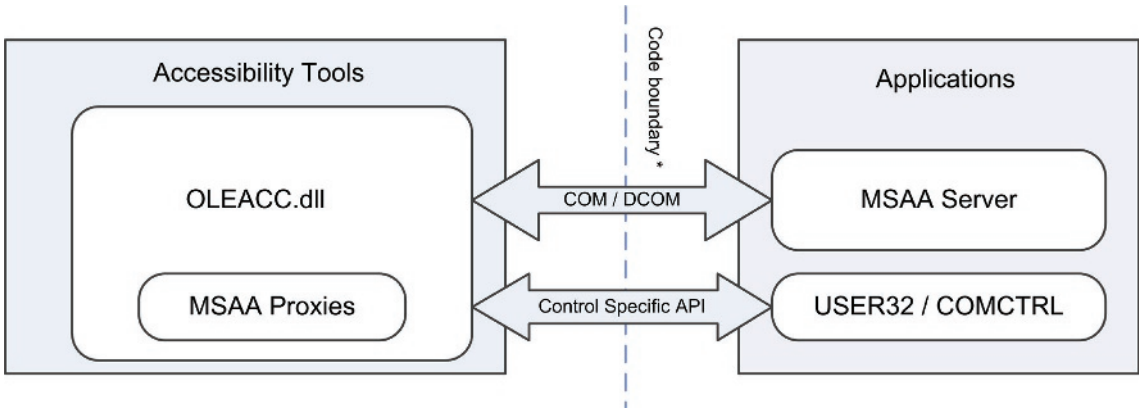
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

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.

3.2 Architecture

The diagrams in this section show the architectures of Microsoft Active Accessibility, UI Automation, and other related implementations. Applications such as word processing programs are called *servers* in Microsoft Active Accessibility and *providers* in UI Automation because they serve or provide information about their user interfaces (UI). Accessibility tools such as screen readers are called *clients* in both Microsoft Active Accessibility and UI Automation because they consume and interact with application UI information. These diagrams provide an overview of each technology and are not intended to present highly detailed views of the architecture and scenarios of Microsoft Active Accessibility, UI Automation, and other implementations discussed in this section.

The system component of the Microsoft Active Accessibility framework, `Oleacc.dll`, aids in the communication between accessibility tools (clients) and applications (servers). The *code boundary* indicates the programmatic boundaries between applications that provide UI accessibility information and accessibility tools that interact with the UI on behalf of users. The boundary can also be a *process boundary* when Microsoft Active Accessibility clients have their own process.



* Also process boundary in case of out-of-process MSAA clients.

Figure 1 — Microsoft Active Accessibility

With UI Automation, the UI Automation Core component (`UIAutomationCore.dll`) is loaded into both the accessibility tools' and applications' processes. This component manages cross-process communication, and it also provides higher level services such as searching for elements by property values.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

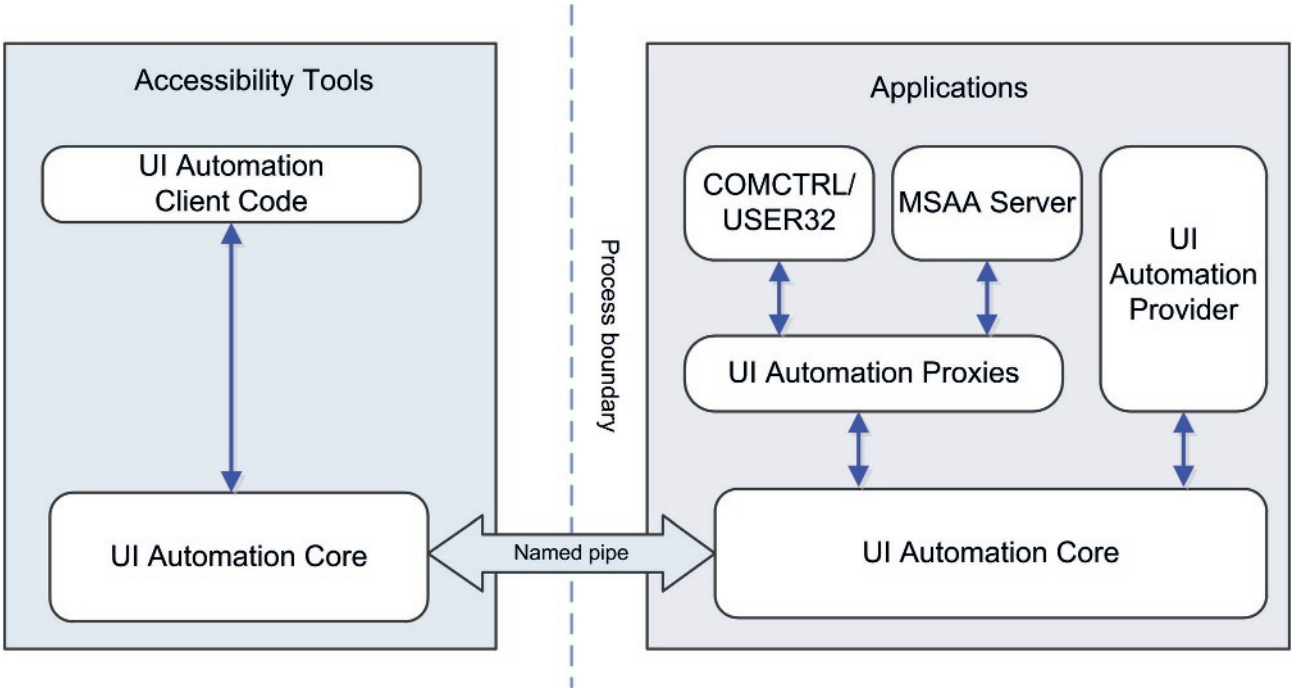


Figure 2 — UI Automation

Using the `IAccessibleEx` interface, applications can improve the accessibility of existing Microsoft Active Accessibility server implementations. Microsoft Active Accessibility server implementations are exposed to clients via the proxy just as regular UI Automation implementations are.

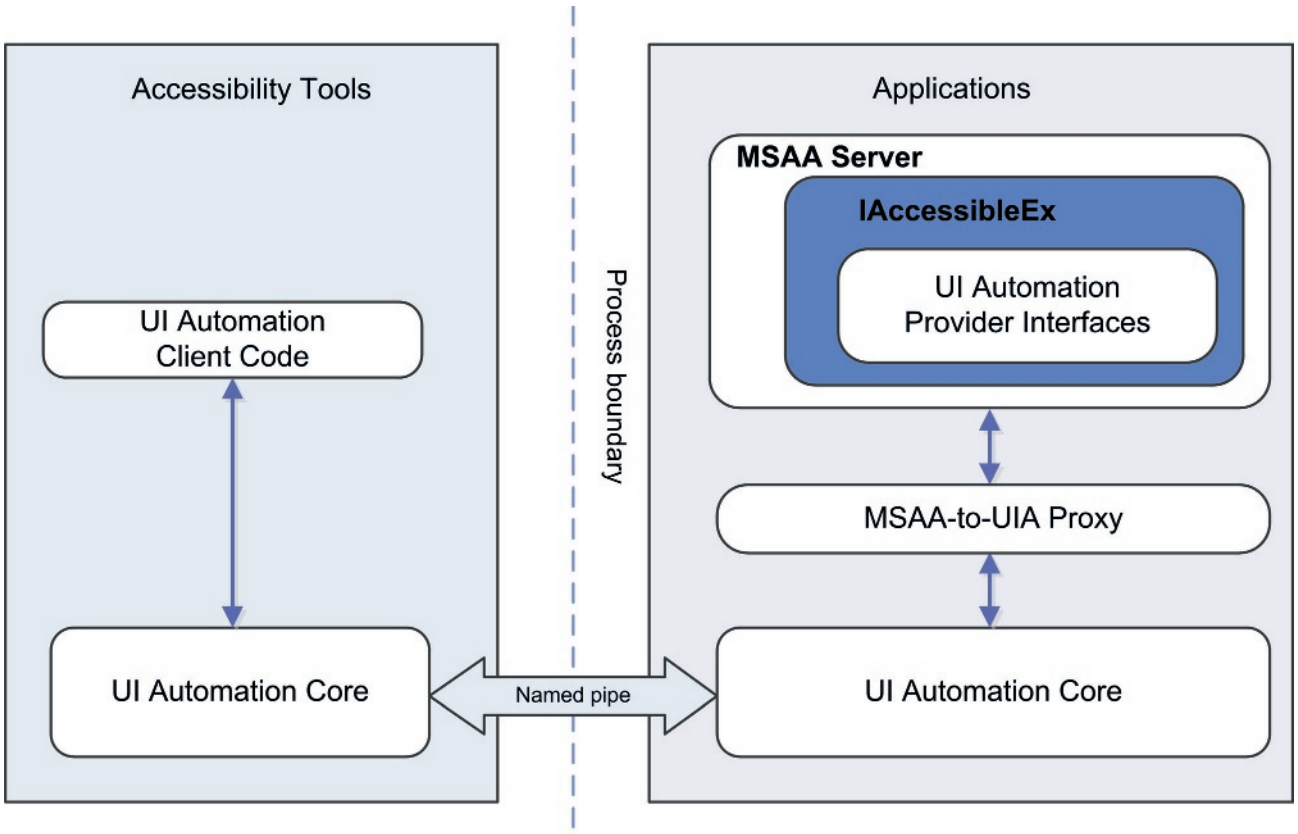
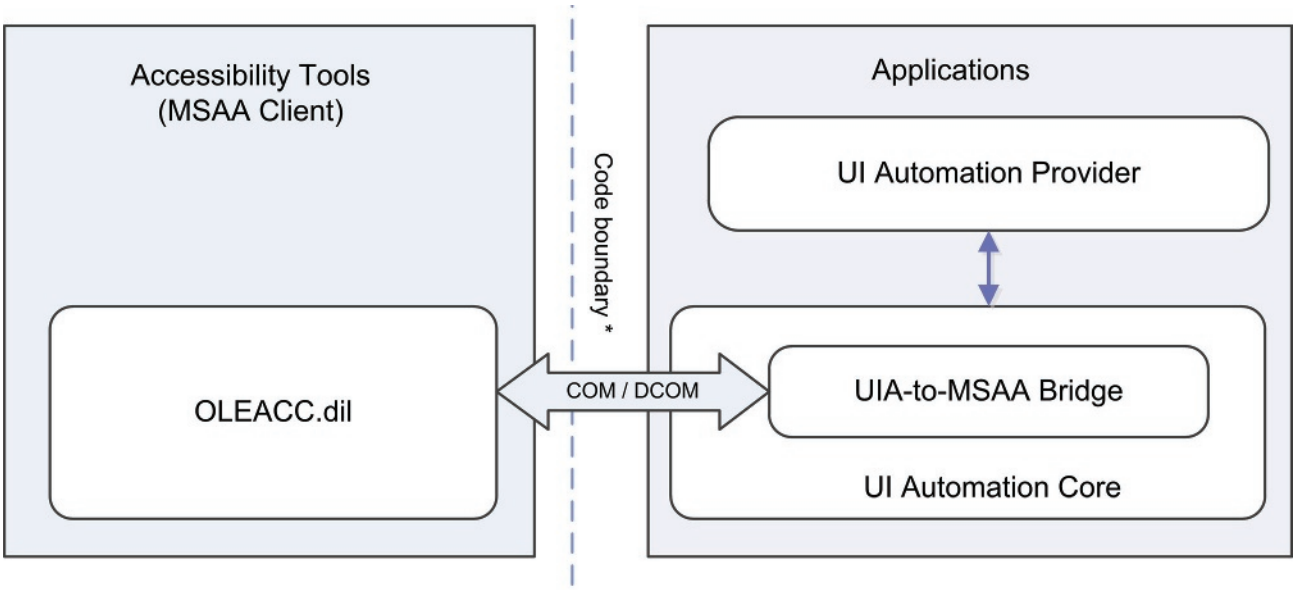


Figure 3 — MSAA-to-UIA Proxy enables UI Automation clients to access Microsoft Active Accessibility servers

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

While the MSAA-to-UIA proxy enables UI Automation clients to access Microsoft Active Accessibility servers, the UIA-to-MSAA bridge does the inverse. It enables Microsoft Active Accessibility clients to access UI Automation providers.



* Also process boundary in case of out-of-process MSAA clients.

Figure 4 — UIA-to-MSAA Bridge enables Microsoft Active Accessibility clients to access UI Automation providers

4 Using the API

4.1 Using the Microsoft Active Accessibility API

In Microsoft Active Accessibility, every UI element is represented by an `IAccessible` interface paired with a `ChildId` value. A UI element represented by such a pair is called an *accessible object*. An accessible object exposes properties, including the object's name, screen location, and other information needed by accessibility aids. The accessible object also provides methods that enable clients to perform an action on the corresponding UI element.

An accessible object that has simple elements associated with it is also called a *parent* or *container*. The parent is represented by a `ChildId` value of `CHILDID_SELF` (or 0 "zero"). The children are represented by a non-zero value (usually a positive sequential number beginning with 1). Child objects that are represented by a non-zero `ChildId` value are called *simple elements*. Simple elements share the same `IAccessible` interface with their parent, but they are differentiated by the `ChildId` value. The `ChildId` assignment is done on a per-instance-of-interface basis, so the IDs must be unique within that context.

For example, a system list box is represented by a proxy object as an accessible object for the overall list box, and simple elements for each list box item. In this case, the accessible object with `CHILDID_SELF` is called a *parent* or *container* of the list items. The individual objects with non-zero `ChildId` values are, on the other hand, called children or list items (contained in the list box).

Simple elements cannot have children of their own. If a UI element has more than two levels of hierarchy, the object representation should be structured in multiple levels of accessible objects instead of parent and simple object pairs.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

4.1.1 Types of Microsoft Active Accessibility support

Microsoft Active Accessibility servers can have two types of support for accessible objects: native and proxied. An application's UI elements determine which type of support is appropriate. Many servers being written today take full advantage of the system provided proxies. They implement Microsoft Active Accessibility only for those custom controls that the system does not proxy.

4.1.1.1 Native Microsoft Active Accessibility implementation

UI elements that implement their own accessible objects are said to provide a *native implementation*. Although the development cost for implementing custom accessible objects can be high, the benefit is complete control over the information exposed to clients. By providing native support, an application is free to innovate in its UI while remaining 100 % accessible.

If an application uses custom controls or other controls that `Oleacc.dll` cannot proxy, a native implementation will need to be provided.

4.1.1.2 Accessible object proxies

Accessible object proxies provide default accessibility information for standard UI elements: USER controls, USER menus, and common controls from `comctl32.dll`. This default support is available from `Oleacc.dll`, and it delivers standard Microsoft Active Accessibility support without additional server development work. However, the application has little control over the information that is exposed.

4.1.2 Retrieving an accessible object

Retrieving an accessible object is the first step to establishing communication between accessibility tools and the target application. Microsoft Active Accessibility clients can initiate this communication by using one of the following `AccessibleObjectFromX` functions provided by `Oleacc.dll`:

Function	Description
`AccessibleObjectFromPoint`	Retrieves an accessible object from a screen coordinate.
`AccessibleObjectFromWindow`	Retrieves an accessible object from a window handle (HWND).
`AccessibleObjectFromEvent`	Retrieves an accessible object from a WinEvent.

4.1.3 The WM_GETOBJECT message

Both Microsoft Active Accessibility and UI Automation send the `WM_GETOBJECT` message to obtain information about an accessible UI object on the desktop. Applications (including accessibility tools or clients) never send this message directly. It is sent only by the accessibility framework (`Oleacc.dll` for Microsoft Active Accessibility or `UIAutomationCore.dll` for UI Automation) in response to calls such as `AccessibleObjectFromPoint`, `AccessibleObjectFromEvent`, or `AccessibleObjectFromWindow`. UI components that support either Microsoft Active Accessibility or UI Automation must handle the message (`WM_GETOBJECT`) correctly.

The return value in response to `WM_GETOBJECT` depends on whether the window or control that receives the message implements Microsoft Active Accessibility, UI Automation, or none of those.

— If implementing Microsoft Active Accessibility for the object and also if the `dwObjID` (`lParam`) was `OBJID_CLIENT`, return the result obtained from `LresultFromObject` function for the `IAccessible` implementation.

— If implementing UI Automation for the object and also if the `dwObjID` (`lParam`) was `UiaRootObjectId`, return an interface to the UI Automation provider using `UiaReturnRawElementProvider` function.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

— If implementing neither Microsoft Active Accessibility nor UI Automation, allow the message to pass to `DefWindowProc`. The accessibility framework will determine if a proxy is available to the particular UI element.

— If `dwObjID` was neither `OBJID_CLIENT` nor `UiaRootObjectId`, allow the message to pass to `DefWindowProc`. The accessibility framework then will process default providers for standard UI elements.

Controls can also use custom values in `dwObjID` to return specific return values or objects to `WM_GETOBJECT`. `OBJID_NATIVEOM` and `OBJID_QUERYCLASSNAMEIDX` can be used for returning a native object model interface or for requesting a specific `Oleacc.dll` proxy.

Microsoft Active Accessibility server and UI Automation provider implementations (the first two implementations in the preceding list) can coexist by handling both `OBJID_CLIENT` and `UiaRootObjectId` accordingly.

Because most Windows common controls and USER controls do not implement either Microsoft Active Accessibility or UI Automation, the UI elements generally don't handle a `WM_GETOBJECT` message. Instead, the accessibility framework (Microsoft Active Accessibility or UI Automation) checks if a proxy object is available for a particular UI element. Otherwise, it will provide to the default proxy object for the host window object.

4.1.4 Special values of Object Identifier

4.1.4.1 Using the OBJID_NATIVEOM to expose a native object model interface

If the UI element supports native object models other than Microsoft Active Accessibility or UI Automation, it can still expose the custom interface by responding to `WM_GETOBJECT` with `OBJID_NATIVEOM` parameter in `dwObjID` (or `lParam`). Use `LresultFromObject` to wrap the interface. Clients can retrieve the interface by calling `AccessibleObjectFromWindow` function with `OBJID_NATIVEOM` as the second parameter.

4.1.4.2 Using the OBJID_QUERYCLASSNAMEIDX to enable certain Oleacc proxy

When Microsoft Active Accessibility sends a `WM_GETOBJECT` message with the `OBJIDQUERYCLASSNAMEIDX` in the object ID, many standard USER or common controls (COMCTL) return one of the following values.

USER or common control	Return value
Listbox	65536+0
Button	65536+2
Static	65536+3
Edit	65536+4
Combobox	65536+5
Scrollbar	65536+10
Status	65536+11
Toolbar	65536+12
Progress	65536+13
Animate	65536+14
Tab	65536+15
Hotkey	65536+16
Header	65536+17
Trackbar	65536+18

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

USER or common control	Return value
Listview	65536+19
Updown	65536+22
ToolTips	65536+24
Treeview	65536+25
RichEdit	65536+28

Generally, only USER and Windows common controls (`comctl32.dll`) return one of the values from the table. If a window returns 0 in response to this message, the window may be one of the following:

— a custom control;

— a control other than one of the controls in the previous table;

— an old version of a system control that doesn't recognize the `WM_GETOBJECT` message.

However, it is possible for the custom control to return a specific return value to enable a USER or COMCTL proxy object by `Oleacc.dll`. Such controls must support all native functions and APIs of the corresponding system control.

4.2 Using the UI Automation API

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, and retrieve information about, UI elements.

4.2.1 UI Automation model

UI Automation exposes every element of the UI to client applications as an "automation element"; that is, 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. 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 behaviour. 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.

The UI Automation client API and the UI Automation core provide services such as tree traversal, searching for elements within the tree, and fetching multiple properties from multiple objects. These services are exposed by the client API, and implemented by the core UI Automation DLL, which runs in-process as necessary to implement these services on behalf of the client.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

4.2.2 UI Automation tree

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 components of the 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 sub-tree. However, providers are not concerned with navigation between these sub-trees. This is managed by the UI Automation core, using information from the default window providers.

UI Automation provides three default views of the UI tree. 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 the `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 from that of a Win32 button. The raw view is obtained by searching for elements without specifying properties or by using `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 dialogue box 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 dialogue box. 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 `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 UI, 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. 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 `ContentViewWalker` to navigate the tree.

4.2.3 UI Automation control patterns, control types, properties, and events

UI Automation support for control patterns, properties, and events goes beyond the support provided by the MSAA `IAccessible` interface.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

4.2.3.1 UI Automation control patterns

A control pattern describes UI attributes and functionality or features of a user interface element. It is an interface with properties and methods.

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

— 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 pattern interfaces provide properties and methods for discovering more functionality about the underlying control. Grouping them together through a special-purpose API (`GetPatternProvider`) distinguishes them from other non-UI centric interfaces.

The UI Automation framework obtains a control pattern interface by using the `GetPatternProvider` function of the `IRawElementProviderSimple` interface. Through the framework, clients can query a control for the control patterns that 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.

The following table provides examples of the functionality represented by different control patterns.

Functionality	Control Pattern
Ability to be checked/unchecked	Toggle
Ability to have numeric value set	RangeValue
Ability to have text value set	Value
Ability to be moved/resized	Transform
Ability to show or hide information	ExpandCollapse

The following table describes the UI Automation control patterns.

Control Pattern	Description
Annotation	— Used to expose the properties of an annotation in a document, for example comments in the margin that are connected to document text.
Dock	Used for controls that can be docked in a docking container. For example, toolbars or tool palettes.
— Drag	— Used for controls that can be docked in a docking container, for example, toolbars or tool palettes.
— DropTarget	— Used to support draggable controls, or controls with draggable items.
ExpandCollapse	Used for controls that can be expanded or collapsed. For example, menu items in an application such as the File menu.
Grid	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.
GridItem	Used for controls that have cells within grids. The individual cells should support the GridItem control pattern; for example, each cell in the "details" view in Microsoft Windows Explorer.

© ISO/IEC 2016 – All rights reserved

AVA Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

Control Pattern	Description
Invoke	Used for controls that can be invoked, such as a button.
ItemContainer	Used for controls that hosts a number of children that may be virtualized.
LegacyIAccessible	Used for controls that have legacy IAccessible implementations, which is supported by a built-in proxy of the UIAutomation-Core.dll
MultipleView	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.
— ObjectModel	— Used to expose a pointer to the underlying object model of a document. This control pattern allows a client to navigate from a UI Automation element into the underlying object model.
RangeValue	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.
Scroll	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.
ScrollItem	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.
Selection	Used for selection container controls; for example, list boxes and combo boxes.
SelectionItem	Used for individual items in selection container controls, such as list boxes and combo boxes.
— Spreadsheet	— Used to expose the contents of a spreadsheet or other grid-based document. Controls that implement the Spreadsheet control pattern should also implement the Grid control pattern.
— SpreadsheetItem	— Used to expose the properties of a cell in a spreadsheet or other grid-based document. Controls that implement the SpreadsheetItem control pattern should also implement the GridItem control pattern.
— Styles	— Used to describe a UI element that has a specific style, fill color, fill pattern, or shape.
SynchronizedInput	Used for UI framework that supports synchronization of input (such as mouse or keyboard input) simulation so that programs can direct the action to specific UI components accurately.
Table	Used for controls that have a grid as well as header information; for example, Microsoft Excel® worksheets.
TableItem	Used for items in a table.
Text	Used for edit controls and documents that expose textual information.
TextEdit	— Used for edit controls that modify text programmatically, for example a control that performs auto-correction or enables input composition.
— TextChild	— Used to access an element's nearest ancestor that supports the Text control pattern.
— TextRange	— Used for retrieving textual content, text attributes, and embedded objects from text-based controls such as edit controls and documents.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

Control Pattern	Description
Toggle	Used for controls in which state can be toggled; for example, check boxes and some menu items that can be selected.
Transform	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.
Value	Used for controls that sustains a data. Value control pattern allows clients to get or set a value (data) on controls that do not support a range of values; for example, an edit control.
VirtualizedItem	Used for a virtualized object in a container that supports the ItemContainer control pattern.
Window	Used for controls that provide fundamental window-based functionality within a tradition graphical UI. For example, a control that supports the Window control pattern can be moved or maximized for the application's window concept.

UI Automation providers implement control patterns to expose the appropriate behaviour for a specific function 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.

4.2.3.2 UI Automation control types

The 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. Each control type has a set of conditions that a control must meet in order to use the `ControlType` property. The conditions include specific guidelines for the UI Automation tree structure, UI Automation property values, control patterns, and UI Automation events.

Having a well-known identifier makes it easier for assistive technology devices to determine what types of controls are available in the UI and how to interact with the controls. The control types included with UI Automation provide a much more comprehensive set of identifiers for indicating controls than Microsoft Active Accessibility `accRole` values.

While control type specifications define a set of requirements and recommendations, applications and controls can add more control patterns or properties while they can still use predefined control types.

The `LocalizedControlType` property is a localized description of the control type, such as "button" for the Button control type. The string can be used by the client application to report the UI information to the users in combination with other UI Automation properties such as Name. For example, screen readers may say "OK button," which can be combination of "OK" from the Name property and "button" from the `LocalizedControlType` property. Controls and applications should never include control type information in the Name property. Otherwise, it will conflict with the control type information.

When `LocalizedControlType` is not specified, the UI Automation framework will provide the default localized string according to the desktop UI language based on the control's `ControlType` property. If a control supplies a known `ControlType` (see the list below), then it does not need to separately implement `LocalizedControlType` because UI Automation will supply the corresponding string automatically.

A custom control can supply unique localized values that may differentiate the element from other controls while the control can still share the same `ControlType` value for the baseline expectations. Required, recommended, or prohibited control patterns and properties are defined by each control type specification.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

The current set of control types consists of the following:

- AppBar
- Button
- Calendar
- Check Box
- Combo Box
- Data Grid
- Data Item
- Document
- Edit
- Group
- Header
- Header Item
- Hyperlink
- Image
- List
- List Item
- Menu
- Menu Bar
- Menu Item
- Pane
- Progress Bar
- Radio Button
- Scroll Bar
- SemanticZoom
- Separator
- Slider
- Spinner
- Split Button
- Status Bar
- Tab
- Tab Item
- Table
- Text
- Thumb
- Title Bar
- Tool Bar
- ToolTip
- Tree Item
- Window

4.2.3.3 UI Automation properties

UI Automation has two main types of properties:

- *Automation element properties* — properties that are applicable to most controls. Examples include `Name`, `Enabled`, and `LabeledBy`. These properties are exposed through `IRawElementProviderSimple.GetPropertyValue` and are of type `AutomationProperty`.
- *Control pattern properties* — properties that are specific to the functionality represented in the different control patterns interfaces. Each control pattern interface exposes a corresponding set of control pattern properties.

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 want to retrieve.

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

4.2.3.4 UI Automation events

UI Automation providers raise events to notify clients of important changes in the UI. Clients register for relevant events and implement event handling methods to receive and process the events when they occur. For more information, see Clause 8.

4.2.3.5 UI Automation provider interfaces

UI Automation provider interfaces can be implemented by managed or native providers. The interfaces expose the same basic information as the Microsoft Active Accessibility `IAccessible` interface (locations, tree structure, focus and hit testing support). Code that implements these interfaces is

© ISO/IEC 2016 – All rights reserved
AVA Documents Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

visible to UI Automation clients, but is also made available as `IAccessible` implementations for in-process or out-of-process Microsoft Active Accessibility clients.

`IRawElementProviderSimple` is the core interface that UI Automation providers use to represent a UI element. It includes methods that expose whether the implementation is a proxy, and methods to expose when an HWND is contained within an element. It also includes the `GetPropertyValue` and `GetPatternProvider` methods that clients use to get property and control pattern values. Providers can also implement the `IRawElementProviderFragment` and `IRawElementProviderFragmentRoot` interfaces to provide additional functionality for complex controls.

4.2.3.6 Custom UI Automation properties, control patterns, and events

The UI Automation framework specification defines a comprehensive set of control patterns, properties, and events. The Microsoft Windows implementation of UI Automation includes the complete set of items defined in the framework specification, and also offers a way to extend the set by registering custom control patterns, properties, and events. In the current implementation for native code, applications are required to register custom UI Automation properties, control patterns, and events before they can be used. For more information, see Clause 11.

4.3 Using the IAccessibleEx interface

The `IAccessibleEx` interface allows existing Microsoft Active Accessibility implementations to add support for UI Automation properties and control patterns. This is done by defining scenarios and specifications for the `IRawElementProviderSimple` and `IAccessibleEx` interfaces that are specific to Microsoft Active Accessibility servers. Existing Microsoft Active Accessibility implementations can take advantage of the work they have already done in creating their existing implementation.

`IRawElementProviderSimple` provides access to UI Automation's control patterns and properties and is used as the interface that represents the UI Element. The `IRawElementProviderSimple` and `IAccessibleEx` interfaces are implemented on the same object, and `QueryInterface` used to go from one to the other.

`IAccessibleEx` handles `IAccessible` bridging issues, such as child IDs, and any other issues in allowing an `IAccessible` element to be treated as an automation element of a UI Automation object.

In summary:

— Existing Microsoft Active Accessibility implementations add support for `IRawElementProviderSimple` (to expose control patterns and properties) and `IAccessibleEx` (to deal with `ChildIds`)

— Existing Microsoft Active Accessibility clients can use the `IRawElementProviderSimple` and `IAccessibleEx` interfaces to access UI Automation properties and control patterns.

— New WinEvents constants are defined to represent events from implementations that are extended by `IAccessibleEx` implementations.

4.3.1 The IAccessibleEx interface implementation

Implementing `IAccessibleEx` is a stepping stone for existing Microsoft Active Accessibility servers to support UI Automation provider interfaces (for example, `IRawElementProviderSimple`). The advantage of this is that existing accessible object implementations are reused, including `IAccessible` properties, the accessible object tree structure, and WinEvents. New code is added only for newly exposed functionality or features.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016

4.3.1.1 Control Patterns: Overlap between Microsoft Active Accessibility and UI Automation

The following control patterns do not exist in Microsoft Active Accessibility, so they can be used in an `IAccessibleEx` implementation.

— `Dock control pattern`

— `Expand Collapse control pattern`

— `Grid control pattern`

— `Grid Item control pattern`

— `Multiple View control pattern`

— `Range Value control pattern`

— `Scroll control pattern`

— `Scroll Item control pattern`

— `Synchronized Input control pattern`

— `Table control pattern`

— `Table Item control pattern`

— `Transform control pattern`

In the case of the RangeValue and Transform control patterns, some methods overlap between the UI Automation (UIA) control pattern and Microsoft Active Accessibility methods. For example, both the `get_accValue` and `put_accValue` methods in MSAA and the RangeValue control pattern `Value` and `SetValue` methods in UIA must be implemented. Internally, an implementation can share code for these Microsoft Active Accessibility and UI Automation methods. The requirement to implement both Microsoft Active Accessibility and UI Automation implementations avoids having a partial implementation of a control pattern interface while keeping the `IAccessible` interface usable by existing Microsoft Active Accessibility clients.

The following UI Automation control patterns are not required when the control has one of the roles outlined below. Otherwise, they should be explicitly supported if relevant.

UI Automation Control Pattern	Microsoft Active Accessibility Role
InvokePattern	ROLE_SYSTEM_PUSHBUTTON, ROLE_SYSTEM_MENUITEM, ROLE_SYSTEM_BUTTONDROPDOWN, ROLE_SYSTEM_SPLITBUTTON, and any other role where `accDefaultAction` is not NULL.
SelectionItemPattern	ROLE_SYSTEM_LISTITEM, ROLE_SYSTEM_RADIOBUTTON
SelectionPattern	ROLE_SYSTEM_LIST
TogglePattern	ROLE_SYSTEM_CHECKBUTTON
ValuePattern	ROLE_SYSTEM_TEXT (when it is not read-only), ROLE_SYSTEM_PROGRESSBAR, ROLE_SYSTEM_COMBOBOX, and any other role when `accValue` is not `NULL`.
WindowPattern	Automatically supported on top-level Win32 HWNDs.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 13066-2:2016