TECHNICAL REPORT

# ISO/IEC TR 13066-3

First edition
2012-09-15

# Information technology — Interoperability with Assistive Technology (AT) —

## Part 3: IAccessible2 accessibility application programming interface (API)

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

*Partie 3: Interface de programmation d'applications (API) d'accessibilité IAccessible2*



Reference number
ISO/IEC TR 13066-3:2012(E)



ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

# Contents

Page

# 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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 13066 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 13066-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 35, *User interfaces*.

ISO/IEC 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)* [Technical Report]

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

# Introduction

Assistive technology (AT) is specialized information technology (IT) hardware or software that is added to or incorporated within a system that increases accessibility for an individual. In other words, it is special purpose IT that interoperates with another IT product enabling a person with a disability to use the IT product.

Interoperability involves the ability to add or replace AT to existing components of IT systems. Interoperability between AT and IT is best facilitated via the use of standardized, public interfaces for all IT components.

This part of ISO/IEC 13066 describes the IAccessible2 API that can be used as a framework to support software to software IT-AT interoperability on the Windows platform.

# Information technology — Interoperability with Assistive Technology (AT) —

## Part 3: IAccessible2 accessibility application programming interface (API)



## 1 Scope

This part of ISO/IEC 13066 provides an overview to the structure and terminology of the IAccessible2 accessibility API.

It provides:

— a description of the overall architecture and terminology of the API;

— further introductory explanations regarding the content and use of the API beyond those found in Annex A of ISO/IEC 13066-1;

— an overview of the main properties, including:

  — of user interface elements,

  — of how to get and set focus,

  — of communication mechanisms in the API;

— a discussion of design considerations for the API (e.g. pointers to external sources of information on accessibility guidance related to using the API);

— information on extending the API (and where this is appropriate);

— an introduction to the programming interface of the API (including pointers to external sources of information).

It provides this information as an introduction to the IAccessible2 API to assist:

— IT system level developers who create custom controls and/or interface to them;

— AT developers involved in programming "hardware to software" and "software to software" interactions.

## 2 Terms and definitions

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

**2.1**
**accessible object**
part of user interface object that is accessible by Microsoft Active Accessibility

NOTE An accessible object is represented by a pair of the `IAccessible` interface and `ChildId` identifier.

**2.2**
**application programming interface**
**API**
collection of invocation methods and associated parameters used by one piece of software to request actions from another piece of software

[ISO/IEC 18012-1:2004, definition 3.1.1]

**2.3**
**application software**
software that is specific to the solution of an application problem

[ISO/IEC 2382-1, definition 10.04.01]

EXAMPLE A spreadsheet program is application software.

**2.4**
**assistive technology**
**AT**
hardware or software added to, or incorporated within, a system that increases accessibility for an individual

EXAMPLE Braille display, screen reader, screen magnification software and eye tracking device.

[ISO 9241-171, definition 3.5]

NOTE Within this part of ISO/IEC 13066, where assistive technology (and its abbreviation AT) is used, it is to be considered as both singular and plural, without distinction. If it is to be used in the singular only, it will be preceded by the article "an" (i.e. an assistive technology). If it is to be used in the plural only, it will be preceded by the adjective "multiple" (i.e. multiple AT).

**2.5**
**client**
component that uses the services of another component

NOTE In this part of ISO/IEC 13066, client refers more specifically to a component that uses the services of either or both Microsoft Active Accessibility and/or UI Automation to access, identify, or manipulate the UI elements of an application.

**2.6**
**Component Object Model**
**COM**
object-oriented programming model that defines how objects interact within a single process or between processes

NOTE In COM, clients have access to an object through interfaces implemented on the object.

**2.7**
**compatibility**
capability of a functional unit to meet the requirements of a specified interface without appreciable modification

[ISO/IEC 2382-1, definition 01.06.11]

**2.8**
**function**
defined objective or characteristic action of a system or component

[IEEE Std. 610.12-1990, unnumbered definition]

EXAMPLE A system has inventory control as its primary function.

**2.9**
**information/communication technology**
**ICT**
technology for gathering, storing, retrieving, processing, analysing and transmitting information

[ISO 9241-20, definition 3.4]

EXAMPLE A computer system is a type of ICT.2.13.

**2.10**
**interface**
shared boundary between two functional units, defined by various characteristics pertaining to the functions, physical interconnections, signal exchanges, and other characteristics, as appropriate

[ISO/IEC 2382-1, definition 01.01.38]

**2.11**
**interoperability**
capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units

[ISO/IEC 2382-1, definition 01.01.47]

**2.12**
**Microsoft Active Accessibility**
COM-based technology that improves the way accessibility aids work with applications running on Microsoft Windows

NOTE It 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.

**2.13**
**operating system**
**OS**
software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input-output control, and data management

NOTE Although operating systems are predominantly software, partial hardware implementations are possible.

[ISO/IEC 2382-1, definition 01.04.08]

**2.14**
**servers**
servers in Microsoft Active Accessibility are components (applications, dlls, etc.) that have UI and expose information about the UI and/or allow it to be manipulated

NOTE The terminology ‘server’ in Microsoft Active Accessibility is a synonym of ‘providers’ in UI Automation.

**2.15**
**service**
functionality made available to a user electronically

[ISO/IEC 24752-1 URC, definition 4.27]

EXAMPLE Airline reservation service, currency translation services, weather forecasting, restaurant recommendations, etc.

**2.16**
**simple element**
<Microsoft Active Accessibility> UI element that shares an `IAccessible` object with other peer elements

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

**2.17**
**software**
all or part of the programs, procedures, rules, and associated documentation of an information processing system

NOTE Software is an intellectual creation that is independent of the medium on which it is recorded.

[ISO/IEC 2382-1, definition 01.01.08]

**2.18**
**system software**
**platform software**
application-independent software that supports the running of application software

[ISO/IEC 2382-1, definition 01.04.02]

EXAMPLE An operating system, a Web browser, or a programming environment (e.g. Java) can be used as a platform for application software.

**2.19**
**user interface**
**UI**
mechanisms by which a person interacts with a computer system

NOTE The user interface provides input mechanisms, allowing users to manipulate a system. It also provides output mechanisms, allowing the system to produce the effects of the users' manipulation.

**2.20**
**user interface element**
**user interface object**
entity of the user interface that is presented to the user by the software

[ISO 9241-171 definition 3.38]

NOTE 1 User interface elements may or may not be interactive.

NOTE 2 Both entities relevant to the task and entities of the user interface are regarded as user interface elements. Different user interface element types are text, graphics and controls. A user interface element may be a representation or an interaction mechanism for a task object (such as a letter, a sales order, electronic parts, or a wiring diagram) or a system object (such as a printer, hard disk, or network connection). It may be possible for the user to directly manipulate some of these user interface elements.

EXAMPLE 1 User interface elements in a graphical user interface include such things as basic objects (such as window title bars, menu items, push buttons, image maps, and editable text fields) or containers (such as windows, grouping boxes, menu bars, menus, groups of mutually-exclusive option buttons, and compound images that are made up of several smaller images).

EXAMPLE 2 User interface elements in an audio user interface include such things as menus, menu items, messages, and action prompts.

EXAMPLE 3 User interface elements in tactile interfaces include such things as tactile dots, tactile bars, surfaces, knobs, and grips.

**2.21**
**WinEvents**
mechanism that allows servers and the Windows operating system to notify clients when an accessible object changes

# 3 General Description

## 3.1 General Description

IAccessible2 was developed by IBM in 2006 to complement Microsoft's earlier work on the Microsoft Active Accessibility (MSAA) API. MSAA provides accessibility services on the Windows® platform. The IAccessible2 set of interfaces allow application developers to leverage their investment in MSAA while also providing assistive technologies (AT), such as screen readers, reliable access to user interface features not previously supported by MSAA. ATs implementing only MSAA must use reverse engineering and heuristic techniques, such as screen scraping, to provide access to features such as rich document editing functions and tables in documents. While such AT implementations can work, they are customized for each application, expensive to develop, frequently unreliable, and often must be reworked when new versions of the applications are released. The additional function IAccessible2 provides over MSAA includes support for rich text, tables, relationships between objects, self-describing actions, application-specific information, and extensible object properties to support Web 2.0 applications.

A major requirement of many information technology (IT) vendors is the ability to efficiently create applications for multiple platforms. IAccessible2 has therefore been harmonized with the UNIX accessibility APIs, Accessibility Toolkit (ATK) and Assistive Technology Service Provider Interface (AT-SPI), to allow for efficient multi-platform development. Adding IAccessible2 events and interfaces to MSAA yields application semantics that are very similar on both Windows and UNIX.

These design principles guided the specification of the IAccessible2 interfaces which are based on the difference between MSAA and the UNIX Accessibility Toolkit, described in ISO/IEC 13066-4. Application vendors can incrementally add IAccessible2 interfaces as needed to their MSAA implementation providing improved accessibility to their application. And Windows assistive technology vendors can keep much of their MSAA design, only adding a check for these new interfaces before using their legacy heuristic code.

Note that IAccessible2 is the name used to refer to the "set" of interfaces and also a specific API within the set of interfaces. In this part of ISO/IEC 13066 "IAccessible2" and "IAccessible2 interfaces" refers to the set of interfaces and "`IAccessible2` interface" refers to the specific API.

## 3.2 Architecture

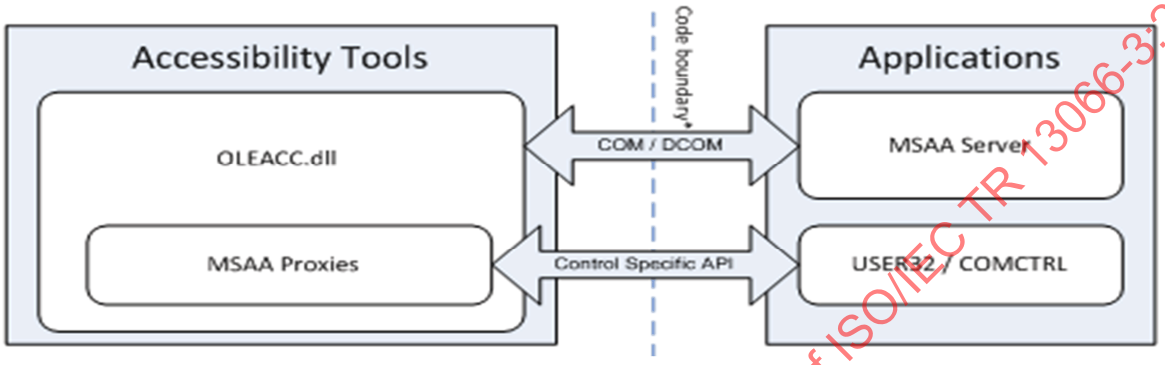
Since IAccessible2 is an extension of MSAA, described in ISO/IEC 13066-2, its architecture matches that of MSAA.

MSAA is based on the Windows Component Object Model (COM), which defines a common way for applications and operating systems to communicate. Applications are considered MSAA servers because they serve or provide information about their user interfaces (UI). Assistive technology products, such as screen readers, are called MSAA clients because they consume and interact with application UI information.

The accessible object is the central interface of Microsoft Active Accessibility and is represented by an `IAccessible` COM interface and an integer `ChildId`. MSAA clients use the interface to access, identify, and manipulate a server's UI and register to be notified of changes to the server UI through event notifications known as WinEvents.

An accessible object can be of two types. A *parent* or *container* object is represented by a ChildID value of `CHILDID_SELF` (or 0 'zero'). The children of a *parent* or *container* object are simple elements that share the same `IAccessible` interface with their parent and are represented by a non-zero ChildID value (usually a positive sequential number beginning with 1). Simple elements cannot have children of their own. Multiple levels of accessible objects are used to represent UI elements that are composed of more than two levels of hierarchy.

As stated in ISO/IEC 13066-2, the system component of the Microsoft Active Accessibility framework, `Oleacc.dll`, aids in the communication between accessibility tools (clients) and applications (servers). The `Oleacc.dll` provides an implementation of the Microsoft Active Accessibility API, the accessibility system framework, and the proxy objects for the Windows Operating System standard controls. The implementation of the Microsoft Accessibility API is an implementation of the IAccessible interface for Windows Operating System controls. Additional implementations of IAccessible, external to Oleacc.dll, are provided by applications running on the Windows Operating System. 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**

Windows application developers and assistive technology vendors are already familiar with MSAA and their COM implementations, and for the information provided, it is reliable and works well. So an important feature of IAccessible2 is that the implementation and semantics of MSAA supporting applications and assistive technologies are preserved. In fact, IAccessible2 builds on the existing COM foundation to expose new interfaces, roles, states, and events.

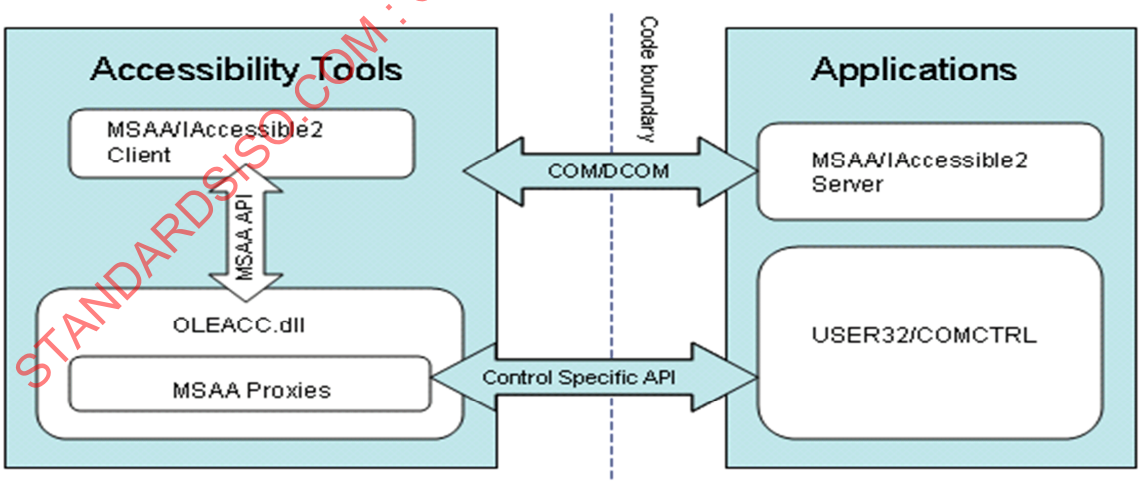



**Figure 2**

Alternatively, an application might choose to have two servers: one for MSAA and one for IAccessible2.

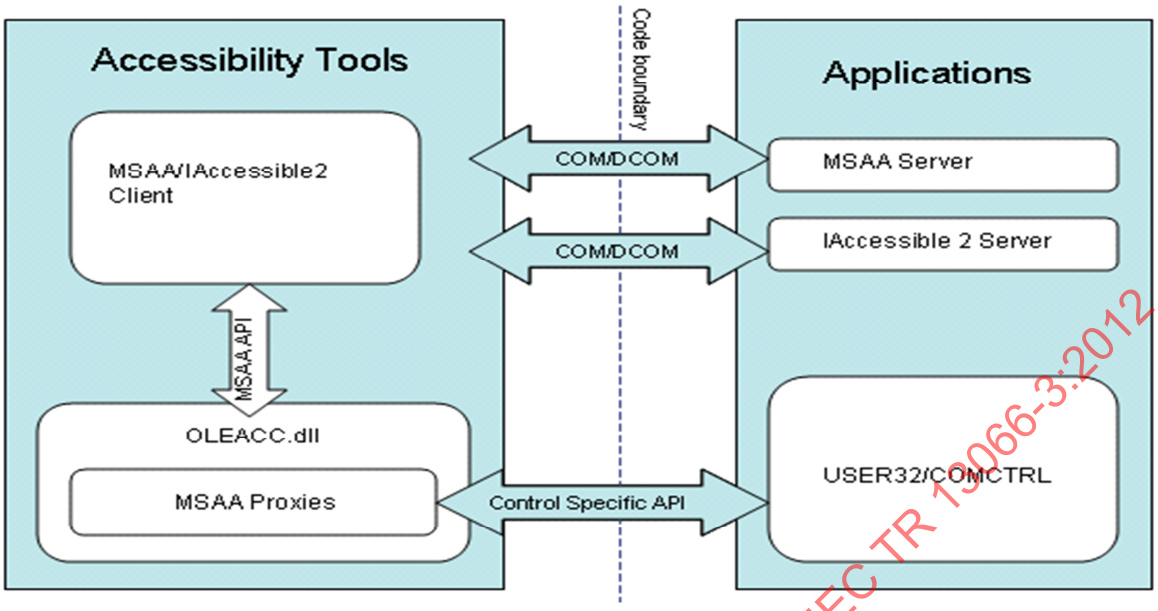



**Figure 3**



# 4 Using the API

## 4.1 Overview

Similar to MSAA, IAccessible2 developers need to identify whether the role of their code is as a *server* -- a provider of IAccessible2 information, or a *client* – a consumer of IAccessible2 information. Typically AT systems are IAccessible2 clients, and query the IAccessible2 servers for information which can then be passed on to the user. In like fashion, servers are usually components in software applications, such as GUI controls. The controls have roles, state and property information, and values.

For example, a screen reader is an IAccessible2 client. It connects to the IAccessible2 server and, using the IAccessible2 APIs, interrogates the server for information about the object, such as its name and value or state, and speaks information about the object to the user in a way that is meaningful. The IAccessible2 server notifies the client of significant changes to the UI, such as focus changes, through WinEvents.

## 4.2 User Interface elements

The IAccessible2 interfaces do not support child IDs, i.e. simple element implementations. Full accessible objects (IAccessible object with ChildID = CHILDID_SELF) must be created for each object that supports IAccessible2. If necessary, some of the IAccessible2 interfaces must also be implemented. Therefore MSAA's `get_accChild` should never return a child ID (other than `CHILDID_SELF`) for an object that implements any of the IAccessible2 interfaces. Although the `IAccessible2` interface is a subclass of MSAA's `IAccessible`, none of the `IAccessible` methods are overridden or extended.

Like MSAA, IAccessible2 objects have roles and states. The `IAccessible2` interface defines an extended set of accessible roles and states in addition to the set of MSAA roles and states. In addition, an IAccessible2 object can have properties describing its relationship to other objects, its position within a group of objects, its locale, and its unique ID.

IAccessible2 provides special interfaces for certain types of accessible objects. These interfaces expose properties that are unique to the particular type of object supported:

— `IAccessibleHyperText` provides the number of links in a document or text object within a document. It will return the link at a particular index.

— `IAccessibleHyperLink` provides information about a specific link or set of links.

— `IAccessibleImage` provides the size, coordinates, and description of an image

— `IAccessibleTable2` provides information about 2-dimensional tables such as the number of rows and columns, the table summary, and the currently selected cells.

— `IAccessibleText` provides information about text such as the total number of text characters, the attributes of the text, the position of the text caret, and the number of non-contiguous selections.

### 4.3 Getting and setting focus

IAccessible2 uses the standard Windows/MSAA mechanisms for getting and setting the keyboard focus. When the focus is on an accessible text object, it is also critical for AT to know the location of the text caret and to be able to set the text caret. MSAA does not provide a mechanism for getting and setting the location of the text caret. Using IAccessible2's `IAccessibleText` interface, a server can provide the location of the text caret and a client can set the position of the text caret.

### 4.4 Communication Mechanisms

Like MSAA, IAccessible2 uses standard Windows interfaces for communication between applications and assistive technologies.

### 4.5 Introduction to Programming interface

#### 4.5.1 COM Interface

The IAccessible2 interface is implemented using the Windows COM model. Servers are implemented as COM objects. Clients call the IAccessible2 objects through the COM interface.

Any programming language which can call a COM interface can query an IAccessible2 server.

## 5 Exposing User Interface Element Information

In ISO/IEC 13066-1, clause 7.1.7 requires that applications

> *provide AT with information about user interface elements, including but not limited to:*
>
> *a) role, state(s), boundary, name, and description of the user interface element*
>
> *b) current value and any minimum or maximum values, if the user interface element represents one of a range of values*
>
> *c) text contents, text attributes, and the boundary of text rendered to the screen.*
>
> *d) the relationship of the user interface element to other user interface elements*
>
> *1) in a single data value, whether this user interface element is a label for another user interface element or is labelled by another user interface element*
>
> *2) in a table, the row and column that it is in, including headers of the row and column if present*

3) *in a hierarchical relationship, any parent containing the user interface element, and any children contained by the user interface element*

Sections 5.1 through 5.4 describe how IAccessible2 supports each of the requirements in subclause 7.1.7.

## 5.1 Role, state(s), boundary, name, and description of the user interface element

IAccessible2 builds on the support provided by the MSAA, adding many new roles including caption, date editor, footer, footnote, form, header, heading, image map, label, and paragraph. For a complete list of roles supported, see AccessibleRoles. The roles are specified using the `role` method on the `IAccessible2` interface. Note that MSAA roles can be exposed through the IAccessible2 `role` method so that assistive technologies don't have to also fetch roles through MSAA `get_accRole`.

In addition, IAccessible2 supports custom roles through the `extendedRole` method.

States are exposed via the `states` method on the `IAccessible2` interface. As with roles, IAccessible2 adds additional states to those provided by MSAA. The additional states include active, armed, invalid entry, required, and selectable text. `IA2_STATE_REQUIRED` and `IA2_STATE_INVALID_ENTRY` are especially useful for creating accessible forms. Applications may use these states to programmatically identify required form fields and indicate where the user has made invalid entries. For a complete list of IAccessible2 states supported see AccessibleStates.

Note that MSAA states can be used with IAccessible2 roles. Likewise, IAccessible2 states can be used with MSAA roles. For example, an object with an MSAA role of `ROLE_SYSTEM_TEXT` can have a state of `IA2_STATE_MULTI_LINE` indicating a multi-line text area. Similarly, an object with an MSAA role of `ROLE_SYSTEM_DOCUMENT` may have a state of `IA2_STATE_EDITABLE` indicating an editable document.

Custom states are supported through the `extendedStates` method.

In addition to the MSAA `IAccessible::get_accLocation` method, boundary information of user interface elements is provided by the `locationInParent` method of the `IAccessibleComponent` interface. For images, the boundary information is provided by a combination of the `imagePosition` and `imageSize` methods of the `IAccessibleImage` interface.

The name and description of most user interface elements are supported via MSAA interfaces. In addition, IAccessible2 defines methods for providing descriptions for images, table columns headers, and table row headers. The `summary` and `caption` methods of the `IAccessibleTable2` interface may also be used for table description information.

For details, see IAccessible2 Interface Reference, IAccessibleComponent Interface Reference, IAccessibleImage Interface Reference, IAccessibleTable2 Interface Reference, and IAccessibleImage Interface Reference.

## 5.2 Current value and any minimum or maximum values, if the user interface element represents one of a range of values

The `IAccessibleValue` interface is implemented by any object that supports a value like progress bars and spin boxes. This interface provides access to the value (`currentValue`) and its upper and lower bounds (`maximumValue` and `minimumValue`). It may be used for objects that have numerical values or text string values.

`IAccessibleValue` also provides a method for setting the value of an object (`setCurrentValue`).

For details, see IAccessibleValue Interface Reference.

## 5.3 Text contents, text attributes, and the boundary of text rendered to the screen

IAccessible2 has two interfaces for text: `IAccessibleText` and `IAccessibleEditableText`. `IAccessibleText` provides methods for obtaining the text value (`text`), attributes (`attributes`), and boundary (`characterExtents`). Additional methods relevant to exposing and tracking the text caret and selection attributes are described in section 7 Keyboard focus.

`IAccessibleEditableText` is used in conjunction with `IAccessibleText` and provides methods to set text attributes and perform clipboard functions such as cut, copy, and paste.

IAccessible2 provides methods to expose embedded links in a document through the `IAccessibleHyperText` interface. Information regarding each individual link to another document is exposed through the `IAccessibleHyperLink` interface which provides information about the associated anchor, the associated link text within the document, and whether the anchor target is still valid.

For details, see IAccessibleText Interface Reference, IAccessibleEditableText Interface Reference, IAccessibleHyperText Interface Reference, and IAccessibleHyperLink Interface Reference and the IAccessible2 Implementation Guide.

## 5.4 The relationship of the user interface element to other user interface elements

The `IAccessibleRelation` interface gives access to an object's set of relations. It provides the type of relation (`relationType`), number of targets (`nTargets`), and the targets themselves (`target`, `targets`). As table structure relationships can be complex, IAccessible2 provides a special interface (`IAccessibleTable2`) for specifying the relationships among table cells.

For details, see IAccessibleRelation Interface Reference and IAccessibleTable2 Interface Reference.

The following sections describe support for the requirements in clause 7.1.7, item d.

### 5.4.1 In a single data value, whether this user interface element is a label for another user interface element or is labelled by another user interface element

In IAccessible2, labels are defined using the role `IA2_ROLE_LABEL`. The `IAccessibleRelation` interface is used to define the relationship (`relationType`) between data objects and their labels. Labels use `relationType` `IA2_RELATION_LABEL_FOR`. And the data objects use `relationType` `IA2_RELATION_LABELLED_BY`.

For details, see IAccessibleRelation Interface Reference.

### 5.4.2 In a table, the row and column that it is in, including headers of the row and column if present

The `IAccessibleTable2` interface provides extensive support for tables. The `rowindex` and `columnindex` methods or the `rowColumnExtentsAtIndex` method can be used to provide the row and column information for a particular cell. The `rowheader` and `columnheader` methods are used to provide the row header and column header for a particular cell.

For details, see IAccessibleTable2 Interface Reference.

### 5.4.3 In a hierarchical relationship, any parent containing the user interface element, and any children contained by the user interface element

Normal parent/child relationships are defined using Windows and MSAA APIs. In cases where this normal parent/child relationship needs to be overridden or augmented, the `IAccessibleRelation` interface may be used. A `relationType` of `IA2_RELATION_NODE_CHILD_OF` indicates that the object is a child of the target object. A `relationType` of `IA2_RELATION_PARENT_WINDOW_OF` indicates that the object is a parent window of the target object.

For details, see IAccessibleRelation Interface Reference.

## 6 Exposing User Interface Element Actions

In ISO/IEC 13066-1, clause 7.1.8 requires software to:

> *programmatically expose a list of available actions on a user interface element and allow assistive technology to programmatically execute any of those actions.*

The `IAccessibleAction` interface provides a method for determining the number of actions (`nActions`), name, description, and keyboard binding (`keyBinding`) if applicable. In addition, a method is provided to perform any of the actions (`doAction`).

For details, see IAccessibleAction Interface Reference.

## 7 Keyboard Focus

In ISO/IEC 13066-1, subclause 7.1.9 requires software to:

> *programmatically expose information necessary to track and modify: focus, text insertion point (where applicable), and selection attributes of user interface elements.*

IAccessible2 builds on the support provided by MSAA for tracking focus as it moves among the user interface objects of the application (`EVENT_OBJECT_FOCUS`).

Once focus is in a rich text field, IAccessible2 events are used to track the movement of the text insertion point (`IA2_EVENT_TEXT_CARET_MOVED`). `IAccessibleText` includes methods for determining the position of the text caret (`caretOffset`) and setting the text caret position (`setCaretOffset`). `IAccessibleText` also includes methods for tracking selected text (`nSelections` and `selection`) and modifying the selection (`setSelection`). Changes to the text selection are indicated by firing the `IA2_TEXT_SELECTION_CHANGED` event.

IAccessible2 provides some special support for the `aria-activedescendant` property of the Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) specification. This property is a bit like the text caret for an accessible text object. The focus is on the parent object, however, the user perceives the focus to be on one of the children of the object with the `aria-activedescendant` property. To support this WAI-ARIA feature, IAccessible2 provides an event called `IA2_EVENT_ACTIVE_DESCENDANT_CHANGED`. This event is fired to notify clients that the active descendant of a WAI-ARIA object has changed.

The `IAccessibleTable2` interface defines methods for identifying which table cells are selected, including entire rows or columns that are selected. It also defines methods for selecting cells, rows, and columns. Changes to selection in a table are indicated by firing the MSAA `EVENT_OBJECT_FOCUS` event.

For more information, see IAccessibleText Interface Reference and IAccessibleTable2 Interface Reference.

## 8 Events

In ISO/IEC 13066-1, clause 7.1.10 requires software to:

> *programmatically expose notification of events relevant to user interactions, including but not limited to:*
>
> *a) changes in the user interface element value;*
>
> *b) changes in the name of the user interface element;*
>
> *c) changes in the description of the user interface element;*
>
> *d) changes in the boundary of the user interface element;*
>
> *e) changes in the hierarchy of the user interface element.*

In most cases, MSAA events are used to notify ATs of significant user interactions. IAccessible2 defines additional events in support of rich text and tables that can be fired using WinEvents, the common communication vehicle for general cross process communication on the Windows desktop. The OEM Reserved range 0x0101-0x01FF is currently allocated for IAccessible2 events. See ISO/IEC 13066-2 section 6.3.4 'The Allocation of WinEvent IDs' for all reserved WinEvent ranges.

For a complete list of events supported by IAccessible2, see AccessibleEventID.

### 8.1 changes in the user interface element value

MSAA events are used to notify ATs of changes in the value of most user interface elements.

IAccessible2 defines additional events for changes to documents, table captions, table column and row headers, table cells including insertions and deletions, table summaries, and text.

### 8.2 changes in the name of the user interface element

MSAA events are used to notify ATs of changes in the name of a user interface element.

### 8.3 changes in the description of the user interface element

MSAA events are used to notify ATs of changes in the description of most user interface elements.

In addition, IAccessible2 defines additional events for changes in the description of table column and row headers.

### 8.4 changes in the boundary of the user interface element

MSAA events are used to notify ATs of changes in the boundary of a user interface element.

### 8.5 changes in the hierarchy of the user interface element

MSAA events are used to notify ATs of changes in the hierarchy of a user interface element.

## 9 Programmatic Modifications of States, Properties, Values, and Text

In ISO/IEC 13066-1, clause 7.1.11 requires software to:

*allow states, properties, values, and text that can be modified by users to be programmatically modified by applications acting as assistive technology.*

MSAA provides methods to allow assistive technology (AT) to modify the value or selection of user interface elements. Iaccessible2 provides the following additional functions to allow AT to modify user interface element attributes and values:

— `IAccessibleTable2` has functions that allow AT to select and unselect rows or columns: `selectRow`, `selectColumn`, `unselectRow`, and `unselectColumn`.

— `IAccessibleText` has functions that allow AT to select and unselect text: `addSelection`, `removeSelection`, and `setSelection`.

— `IAccessibleEditableText` has functions that allow AT to insert, delete, and set the properties of text: `deleteText`, `insertText`, `cutText`, `pasteText`, `replaceText`, and `setAttributes`.

— `IAccessibleValue` has a function that allows AT to modify the value of a user interface element: `setCurrentValue`.

For details, see IAccessibleTable2 Interface Reference, IAccessibleText Interface Reference, IAccessibleEditableText Interface Reference, and IAccessibleValue Interface Reference.

# 10 Design Considerations

## 10.1 Using IA2

### 10.1.1 The IAccessible2Proxy.dll

Applications building IAccessible2 servers and clients need to provide a proxy DLL, but because IAccessible2 is a standard, the proxy DLL is best managed external to the application by a third party. The convention is for ATs to manage the DLL, and place and register it on the system at install time; leaving it on the system should the AT be uninstalled.

An IAccessible2 COM proxy stub DLL is available on the Linux Foundation IAccessible2 site, as well as instructions for building it directly.

### 10.1.2 Using IAccessibleApplication to get Application's name and version

`IAccessibleApplication` is used to retrieve the application's name, version, the toolkit or bridge name, and the toolkit or bridge version.

### 10.1.3 Discovering Interfaces and Services

In general AT should try IAccessible2 interfaces, followed by using the MSAA interfaces. (In cases where an application is known to have custom interfaces which provide information not supplied by IAccessible2 or MSAA, then those custom interfaces can be used.) The AT can then, by default, support unknown IAccessible2/MSAA applications, without the application developers having to request AT vendors for support on an individual application by application basis.

The `QueryInterface` method of the `IUnknown` interface provides information to the client about which interfaces are implemented by the MSAA server. ATs typically use the `QueryInterface` method when switching between any of the `IAccessible` interfaces.

However, because MSAA 2.0 causes clients to talk to a server's `IAccessible` interface through an intermediate MSAA-provided wrapper, the `QueryService` method of the `IServiceProvider` interface is more conveniently implemented by servers because it can be implemented on the same or a separate object without the requirements of symmetry, transitivity, or preserving object identity. The `QueryService` method must be used to switch back and forth between a reference to an MSAA `IAccessible` interface and an IAccessible2 based interface.

It is typical for applications implementing IAccessible2 to implement all applicable IAccessible2 interfaces on a single object. It is possible, however, that an application could use more than one object to implement the full set of IAccessible2 interfaces and as a result, the `QueryService` method must be used. ATs that want to allow for access to the widest range of existing and future applications should use the `QueryService` method rather than the `QueryInterface` method to discover interfaces. If the interface being switched to is implemented on the same object there should not be a noticeable difference in performance and if the interface is implemented on another object the application can transfer the request to the secondary object saving exception handling code in the AT.

Two methods are used under different circumstances to switch between interfaces: `QueryInterface` and `QueryService`.

`QueryInterface` is a method of the `IUnknown` interface. It provides information to the client about which interfaces are implemented by the MSAA server. ATs typically use the `QueryInterface` method when switching between any of the IAccessible interfaces.

`QueryService` is a method of the `IServiceProvider` interface. It must be used to switch back and forth between a reference to an MSAA `IAccessible` interface and an IAccessible2 based interface. `QueryService` is more conveniently implemented by servers because it can be implemented on the same or a separate object without the requirements of symmetry, transitivity, or preserving object identity.

`QueryService` should be used rather than `QueryInterface` to discover interfaces, because although applications typically implement all applicable IAccessible2 interfaces on a single object, it is possible for an application to use more than one object to implement the full set of IAccessible2 interfaces. Using `QueryService` to discover interfaces allows ATs to provide access to the widest range of existing and future applications. When the full set of IAccessible2 interfaces are implemented on the same object there is not typically a noticeable difference in performance during switching. When interfaces are implemented on multiple objects the application can transfer the request to a secondary object saving exception handling code in the AT.

```
// Example of retrieving the IAccessible2 interface
// First get the IServiceProvider interface, get IAccessible2 from that
IAccessible* pAcc = NULL;
IAccessible2* pAcc2 = NULL;
VARIANT varChild;
POINT point;

HRESULT hr = AccessibleObjectFromPoint(point, &pAcc, &varChild);
if (SUCCEEDED(hr) && pAcc && (varChild.vt == VT_I4)) {
    IServiceProvider *pProv = NULL;
    HRESULT res = pAcc->QueryInterface(IID_IServiceProvider,
                                       (void**)&pProv);

    if(SUCCEEDED(res) && pProv){
        res = pProv->QueryService(IID_IAccessible,IID_IAccessible2,
                                  (void**)&pAcc2);
    }
    else{
        pAcc2 = NULL;
    }
}
```

### 10.1.4 Component – building block of widgets

`IAccessibleComponent` provides information about the location of the component in the parent's space, as well as foreground and background colors. Implementations should consider that the component may be hidden or be virtual without set bounds.

```
// Get the location in parent and the background and foreground color
IAccessibleComponent* pAccessibleComponent = NULL;
HRESULT res, result;
res = pAccessible2->QueryInterface
               (IID_IAccessibleComponent, (void**) &pAccessibleComponent);
if(SUCCEEDED(res) && pAccessibleComponent)
{
    long locationX = 0L;
    long locationY = 0L;
    long foreground = 0L;
    long background = 0L;
    res = pAccessibleComponent->get_locationInParent
                                                    (&locationX,&locationY);
```

```
    if(SUCCEEDED(res) && locationX >= 0L && locationY >= 0L)
    {
        locationX +=1L;
    }
    HRESULT res1, res2;
    res1 = pAccessibleComponent->get_background (&background);
    res2 = pAccessibleComponent->get_foreground (&foreground);
    if(SUCCEEDED(res1) && SUCCEEDED(res2))
    {
        // Check that contrast is sufficient, else return failure
        long bright = Brightness(foreground, background);
        if (bright < 125)
        {
            result = E_FAIL;
        }
    }
}
// Method Brightness compares the relative contrast of two RGB values.
//              Formula:              http://www.splitbrain.org/blog/2008-09/18-
calculating_color_contrast_with_php
long TestAcc::Brightness(long rgb1, long rgb2){
    long br1 = (299*GetRValue(rgb1) + 587*GetGValue(rgb1)
                                    + 114*GetBValue(rgb1) ) / 1000;
    long br2 = (299*GetRValue(rgb2) + 587*GetGValue(rgb2)
                                    + 114*GetBValue(rgb2) ) / 1000;
    return abs(br1-br2);
}
```

### 10.1.5 Discovering actions on accessible objects

`IAccessibleAction` provides information on actions that can be requested on the accessible objects. Example actions might include delete, activate, or provide help. The actions can be discovered and relayed to the user for information or selection.

```
// Example steps for discovery of actions
HRESULT res = pAccessible2->QueryInterface
                    (IID_IAccessibleAction,(void**)&pAccessibleAction);
if(SUCCEEDED(res) && pAccessibleAction){
    long nActions = 0L;
    res = pAccessibleAction->nActions(&nActions);
    if(SUCCEEDED(res)){
        BSTR name = NULL;
        BSTR description = NULL;
        // For each action, get the name, description, and bindings
        for (int iAction=0 ; iAction<nActions; iAction++){
            res = pAccessibleAction->get_name(iAction, &name);
            if(SUCCEEDED(res) && name){
                Process(name);
                ::SysFreeString(name);
            }
            res = pAccessibleAction->get_description
                                                    (iAction, &description);
            if(SUCCEEDED(res) && description){
                Process(description);
                ::SysFreeString(description);
            }

            BSTR *aKeyBindings;
            long aNumMaxBinding = 10;
```

```
            long aNumBindings = 0;
            res = pAccessibleAction->get_keyBinding
                        (iAction, aNumMaxBinding, &aKeyBindings, &aNumBindings);
            if(SUCCEEDED(res)){
                for (int jBind=0; jBind<aNumBindings; jBind++){
                    Process(aKeyBindings[jBind]);
                    ::SysFreeString(aKeyBindings[jBind]);
                }
                // Free array of BSTRs, allocated with CoTaskMemAlloc
                CoTaskMemFree(aKeyBindings);
            }
        }
    }
}
```

### 10.1.6 Working with images

The `IAccessibleImage` interface is simple and not always implemented. It is primarily used when the image is part of editable content or when the description is needed to define a behavior and the image itself has significant information to display as well. The image location may be queried with reference to either the parent object or the entire screen.

```
// Get the location, size, and description of the image
IAccessibleImage* pAccessibleImage = NULL;
res = pAccessible2->QueryInterface
                        (IID_IAccessibleImage, (void**) &pAccessibleImage);
if(SUCCEEDED(res) && pAccessibleImage)
{
    enum IA2CoordinateType coordinateType = IA2_COORDTYPE_SCREEN_RELATIVE;
    long locationX = 0L;
    long locationY = 0L;
    res = pAccessibleImage->get_imagePosition
                                    (coordinateType,&locationX,&locationY);
    if(SUCCEEDED(res)){
        SetLocation(locationX, locationY);
    }

    long height = 0L;
    long width = 0L;
    res = pAccessibleImage->get_imageSize(&height,&width);
    if(SUCCEEDED(res)){
        SetImageSize(height,width);
    }

    BSTR description;
    res = pAccessibleImage->get_description(&description);
    if(SUCCEEDED(res)){
        Process(description);
        ::SysFreeString(description);
    }
}
```

### 10.1.7 Working with number values

Number values are used when there is an object that has a minimum, maximum and a current value. A current value set outside of the minimum and maximum is clipped to closest of the minimum or maximum. The returned value type depends on the implementation. `get_currentValue` may return a different type than the minimum or maximum.

### 10.1.8 Working with tables

The `IAccessibleTable2` interface works with two-dimensional tables.

Typically all accessible objects that represent cells or cell-clusters of a table will be at the same time children of the table. In this case `IAccessible::indexInParent` will return the child index which then can be used when calling `IAccessibleTable2::rowIndex` and `IAccessibleTable2::columnIndex`.

In some cases that kind of implementation will not be possible. When the table cells are not direct children of a table, the object representing the cell can define a "`table-cell-index`" object attribute identifying the 0-based table cell index. This object attribute is obtained by parsing the attribute string returned by `IAccessible2::attributes`. The "`table-cell-index`" attribute can be used just like a child index of the typical case. ATs should first test for the presence of the "`table-cell-index`" attribute and if it is not present then `IAccessible2::indexInParent` can be used as in the typical case where cells are direct children of the table.

```
// Example of selecting and reading the second and third columns in a table
long nColumns = 0L;
HRESULT res = pAccessible2->QueryInterface
                        (IID_IAccessibleTable2,(void**)&pAccessibleTable2);
if(SUCCEEDED(res) && pAccessibleTable2){
   res = pAccessibleTable2->get_nColumns(&nColumns);
   if (SUCCEEDED(res) && nColumns >= 3)
   {
      // Select and speak the second and third column
      for (long k=1; k<3 ; k++)
      {
         BSTR cdescription = NULL;
         long rowCount = 0L;
         HRESULT res1 = pAccessibleTable2->selectColumn(k);
         HRESULT res2 = pAccessibleTable2->get_columnDescription
                                                        (k,&cdescription);
         if(SUCCEEDED(res2) && cdescription){
            Process(cdescription); // Speak description
            ::SysFreeString(cdescription);
         }
         res = pAccessibleTable2->get_nRows(&rowCount);
         if(SUCCEEDED(res) && rowCount){
            for (long m=1; m<rowCount; m++) // For each row
            {
               IAccessible2* spIA2 = NULL;
               IUnknown*  spIUnk = NULL;
               // The accessible comes back as IUnknown
               res = pAccessibleTable2->get_cellAt(m, k, &spIUnk);
               if(SUCCEEDED(res) && spIUnk){
                  IServiceProvider *pProv2 = NULL;
                  res = spIUnk->QueryInterface
                                    (IID_IServiceProvider, (void**) &pProv2);
                  if(SUCCEEDED(res) && pProv2){
                     // Use service provider to get the IAccessible2 interface
                     res = pProv2->QueryService
                             (IID_IAccessible,IID_IAccessible2,(void**)&spIA2);
                     if(SUCCEEDED(res) && spIA2){
                        // Use a variant with CHILDID_SELF to request value
                        BSTR pszValue = NULL;
                        VARIANT varChild ;
                        varChild.lVal = CHILDID_SELF;
                        varChild.vt = VT_I4;
                        HRESULT res = spIA2->get_accValue(varChild, &pszValue);
```

```
                        if(SUCCEEDED(res) && pszValue){
                          Process(pszValue);
                          ::SysFreeString(pszValue);
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
    }
}
```

### 10.1.9 Working with text

#### 10.1.9.1 Offset parameters

The interfaces that deal with text have methods that specify a `startOffset` and `endOffset`. The substring parameters are specified as follows: If `startOffset` is less than `endOffset`, the substring starts with the character at `startOffset` and ends with the character just before `endOffset`. If `endOffset` is lower than `startOffset`, the result is the same as a call with the two arguments exchanged. If both indices have the same value, an empty string is defined. The whole text can be defined by passing the indices zero and `IAccessibleText::nCharacters`.

#### 10.1.9.2 Read only text

The text, caret position and currently selected text are all independently changeable. Multiple simultaneous text selections are supported, and may be individually referenced. Only one caret position is supported.

```
// Example of responding to a text selection
IAccessibleText* pAccessibleText = NULL;
res       =       pAccessible2->QueryInterface(IID_IAccessibleText,       (void**)
&pAccessibleText);
if(SUCCEEDED(res) && pAccessibleText)
{
    long nSelections = 0L;
    res = pAccessibleText->get_nSelections(&nSelections);
    if(SUCCEEDED(res))
    {
        long startOffset = 0L;
        long endOffset = 0L;
        for (long iSel=0; iSel<nSelections; iSel++)
        {// For each selection
            res = pAccessibleText->get_selection
                                            (iSel,&startOffset,&endOffset);
            // Validate return values
            if(SUCCEEDED(res)
                        && startOffset >= 0L
                        && endOffset >= 0L
                        && startOffset < endOffset)
            {
                BSTR text;
                res = pAccessibleText->get_text
                                            (startOffset, endOffset, &text);
                if(SUCCEEDED(res))
                {
```