TECHNICAL REPORT

ISO/IEC TR 13066-4

First edition
2015-11-01

Information technology — Interoperability with assistive technology (AT) —

Part 4: Linux/UNIX graphical environments accessibility API

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

Partie 4: Accessibilité API des environnements graphiques lInux/UNIX

Reference number
ISO/IEC TR 13066-4:2015(E)

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

Contents

Page

Foreword ..... **iv**

Introduction ..... **v**

1 Scope **1**

2 Terms and definitions **1**

3 Overview **4**
3.1 General description 4
3.2 Architecture 5
3.2.1 ATK Aware Toolkits 6
3.2.2 AT-SPI Aware Assistive Technologies 11
3.3 Support apart from AT-SPI/ATK 12

4 Using the API **12**
4.1 Overview 12
4.2 User Interface elements 12
4.3 Getting and setting focus 13
4.4 Communication mechanisms 14
4.5 How GNOME uses the ATK/AT-SPI accessibility Application Programming Interface 14

5 Exposing User Interface Element Information **15**
5.1 Role, state(s), boundary, name, and description of the user interface element 15
5.2 Current value and any minimum or maximum values, if the user interface element represents one of a range of values 16
5.3 Text contents, text attributes, and the boundary of text rendered to the screen 17
5.4 The location of the user interface element in relation to other user interface elements 17

6 Exposing User Interface Element Actions **18**

7 Keyboard focus **18**

8 Events **19**
8.1 Changes in the user interface element value 19
8.2 Changes in the name of the user interface element 20
8.3 Changes in the description of the user interface element 20
8.4 Changes in the boundary of the user interface element 20

9 Programmatic modifications of states, properties, values, and text **20**

10 Design considerations **21**
10.1 Using AT-SPI/ATK 21

11 Further information **21**
11.1 Testing Accessibility with Accerciser 22

Bibliography **23**

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*.

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)* [Technical Report]

— *Part 4: Linux/UNIX graphical environments accessibility API* [Technical Report]

— *Part 6: Java accessibility application programming interface (API)* [Technical Report]

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 Assistive Technology (AT) to existing components of Information Technology (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 following.

AT-SPI

The Assistive Technology Service Provider Interface (AT-SPI) API, which can be used as a toolkit agnostic framework to support software to software IT-AT interoperability on Linux and UNIX graphical desktop environments.

ATK

The Accessibility Toolkit (ATK) library provides a set of interfaces in support of AT-SPI on the GUI application side. The interfaces are toolkit-independent implementations could be written for any widget set, such as GTK, Motif, or Qt.

Information technology — Interoperability with assistive technology (AT) —

Part 4:
Linux/UNIX graphical environments accessibility API

1 Scope

This part of ISO/IEC 13066 provides an overview to the structure and terminology of the Linux/UNIX graphical environments accessibility API.

It will provide the following:

— 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 ISO/IEC 13066-1:2011, Annex A;

— an overview of the main properties, including

  — of user interface elements,

  — of how to get and set focus, and

  — 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 will provide this information as an introduction to the Java API to assist the following:

— 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
A11Y
short form of “accessibility” or of “accessible”

2.2
accessible object
part of the *user interface* (2.21) that is accessible by and exposes the Java accessibility API

Note 1 to entry: An accessible object is represented by an object of the “AccessibleContext” Java class.

2.3
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

2.4
application software
software that is specific to the solution of an application problem

EXAMPLE A spreadsheet program is application software.

2.5
assistive technology
AT
hardware or software that is added to or incorporated within a system that increases accessibility for an individual

EXAMPLE Braille displays, screen readers, screen magnification software, and eye tracking devices are assistive technologies.

[SOURCE: ISO 9241-171:2008, 3.5]

Note 1 to entry: Within this part of ISO/IEC 13066, where Assistive Technology (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).

2.6
ATK
The Accessibility Toolkit
library which describes a set of interfaces that support the AT-SPI on the GUI application side

Note 1 to entry: The ATK interfaces are toolkit-independent - implementations could be written for any widget set, such as GTK, Motif or Qt.

2.7
AtkObject
base object class for the Accessibility Toolkit API

Note 1 to entry: This is the ATK analog of MSAA's and IAccessible2's "accessible object". The AtkObject structure is not accessed directly.

2.8
AT-SPI
The Assistive Technology Service Provider Interface
API that can be used as a toolkit agnostic framework to support software to software IT-AT interoperability on Linux and UNIX graphical desktop environments

2.9
clients
components that use 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 either or both AT-SPI and/or ATK to access, identify, or manipulate the UI elements of an application.

2.10
daemon
software application that is not invoked explicitly, but lies dormant waiting for some condition(s) to occur

2.11
embedded device
embedded system
computer system designed to perform one or a few dedicated functions often with real-time computing constraints

Note 1 to entry: It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Note 2 to entry: In general, "embedded system" is not a strictly definable term, as most systems have some element of extensibility or programmability, e.g. hand-held computers share some elements with embedded systems such as the operating systems and microprocessors which power them, but they allow different applications to be loaded and peripherals to be connected. Moreover, even systems which don't expose programmability as a primary feature generally need to support software updates. On a continuum from "general purpose" to "embedded," large application systems will have subcomponents at most points even if the system as a whole is "designed to perform one or a few dedicated functions," and is thus appropriate to call "embedded."

2.12
function
defined objective or characteristic action of a system or component, e.g. a system has inventory control as its primary function

[SOURCE: IEEE Std. 610.12-1990]

2.13
interface
shared boundary between two functional units, defined by various characteristics pertaining to the functions, physical interconnections, signal exchanges, and other characteristics, as appropriate

2.14
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

2.15
inter-process communication
IPC
mechanism by which different software processes communicate with each other – across process boundaries, runtime environments, and sometimes also computers and *operating systems* (2.16)

2.16
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 1 to entry: Although operating systems are predominantly software, partial hardware implementations are possible.

2.17
servers
in the context of this part of ISO/IEC 13066 and of assistive technology, servers are components (applications, libraries, etc.) that have UI and expose information about the UI and/or allow it to be manipulated

2.18
service
functionality made available to a user electronically, e.g. airline reservation service, currency translation services, weather forecasting, restaurant recommendations are all services

[SOURCE: ISO/IEC 24752-1:2014, 4.27]

2.19
software
all or part of the programs, procedures, rules, and associated documentation of an information processing system

Note 1 to entry: Software is an intellectual creation that is independent of the medium on which it is recorded.

2.20
system software
platform software
application-independent software that supports the running of *application software* (2.4), e.g. an *operating system* (2.16), a Web browser, or a programming environment – Java can be used as a platform for application software

2.21
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. It also provides output mechanisms, allowing the system to produce the effects of the users' manipulation.

2.22
user interface element
user interface object
user interface component
entity of the user interface that is presented to the user by the software

[SOURCE: ISO 9241-171:2008, 3.38]

Note 1 to entry: User interface elements may or may not be interactive.

Note 2 to entry: 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.

3 Overview

3.1 General description

AT-SPI/ATK was originally developed by Sun Microsystems for the GNOME platform, a common and accessible graphical desktop for Linux / UNIX graphical environments. GNOME is an open source project delivering a collection of software libraries and applications. It was formerly rationalized as an acronym meaning GNU Network Object Model Environment. From the beginning, AT-SPI/ATK has served as a platform-neutral, toolkit agnostic framework for providing bi-directional communication between AT and applications. Through the use of AT-SPI, an application's components' state, property, and role information is communicated directly to the end user's AT, thereby facilitating bi-directional (input and output) user interactivity with, and control over, an application or compound document instance.

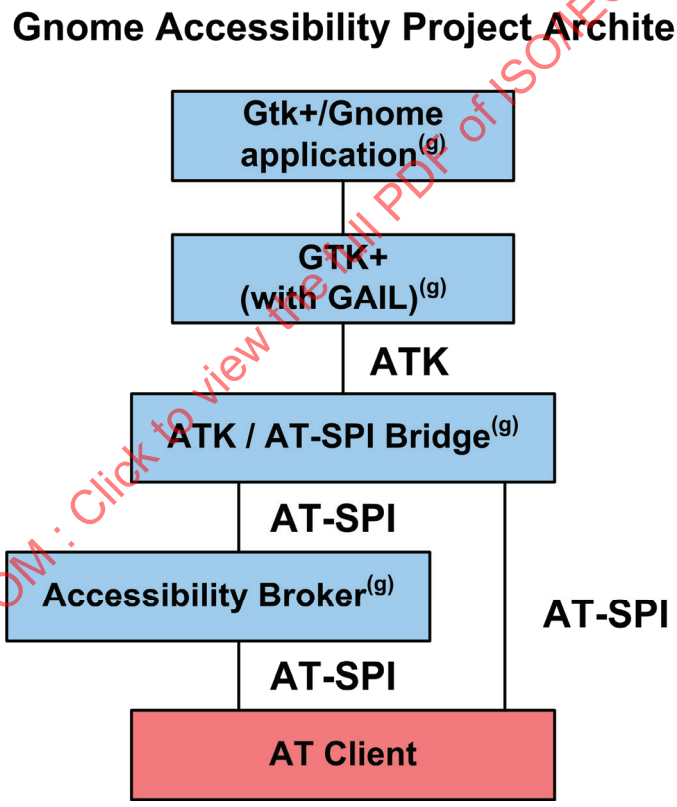
It includes support for rich text, tables, and relationships between objects, self-describing actions, application-specific information, and extensible object properties to support Web 2.0 applications.

As of this writing, both the GNOME and KDE platforms directly support AT-SPI/ATK on Linux/UNIX graphical desktops. For applications, support is available through several specific user interface toolkits, including GTK+ (a library for creating graphical user interfaces which works on many UNIX-like platforms, Windows, and on framebuffer devices), UNO, XUL, and Java/Swing (as described in ISO/IEC TR 13066-6). It can also be achieved by implementing support for ATK or the Java Accessibility API directly, or by AT-SPI directly.

3.2 Architecture

Accessibility support on Linux/UNIX graphical desktops implements the familiar client-server paradigm. The GNOME Accessibility Architecture provided on most Linux and UNIX graphical desktops -the only accessibility architecture available for Linux and UNIX graphical desktops as of this writing- distinguishes among ATK aware applications, assistive technologies and an accessibility broker.

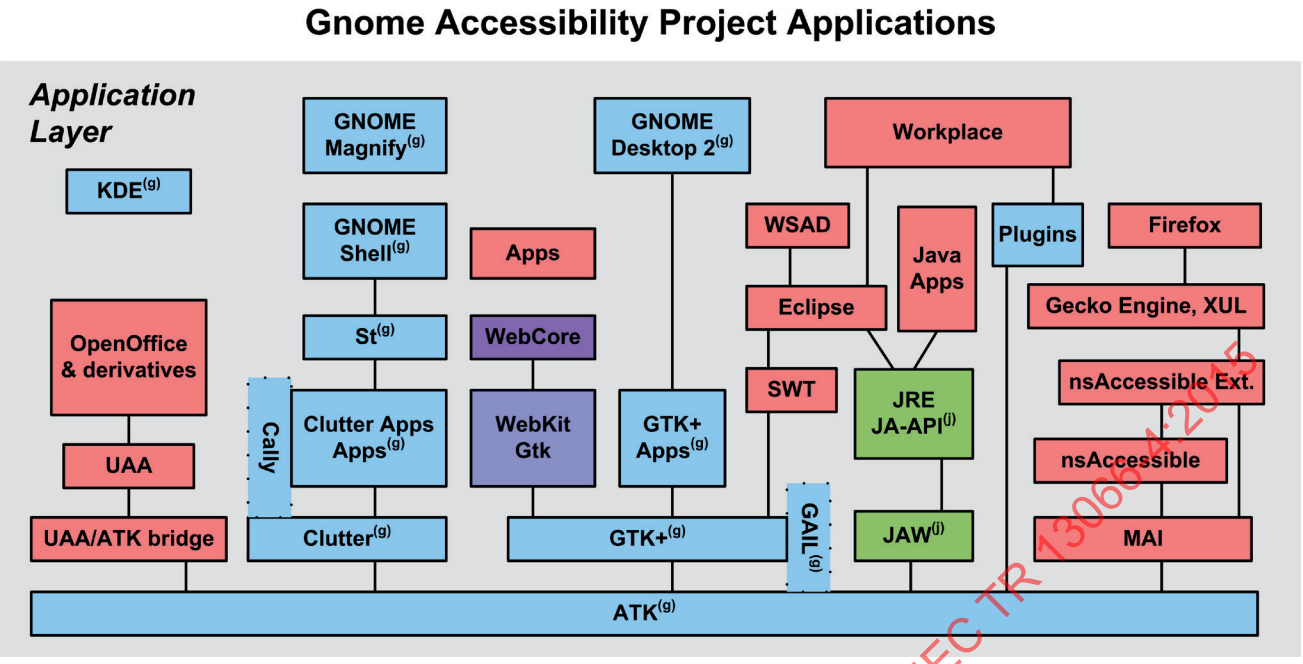
Communications between applications and assistive technologies (AT) is achieved using AT-SPI, which facilitates communications with ATK aware applications on the one hand, and with AT on the other. Generally, only AT is directly aware of AT-SPI.



Key

(g) GNOME

Figure 1 — AT-SPI/ATK Architecture Overview



Key

(g) GNOME

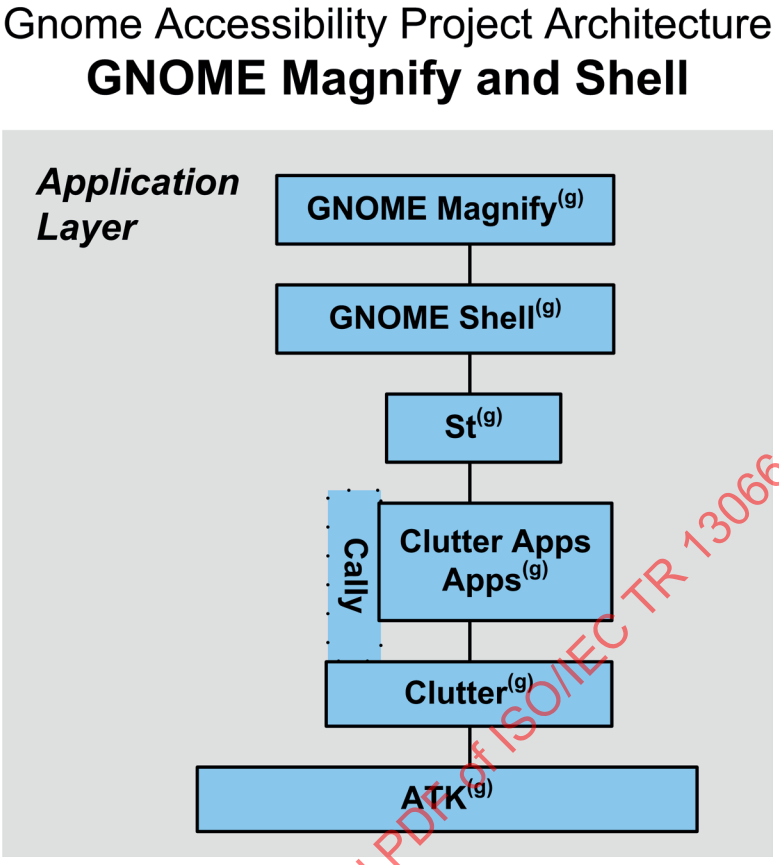
(j) Java; everything else is Application

Figure 2 — Application Toolkits Supporting AT-SPI/ATK

3.2.1 ATK Aware Toolkits

ATK aware applications are applications that offer information about their user interface via the AT-SPI protocol. This can be achieved in several ways.

— GNOME applications get AT-SPI support through the GTK+ toolkit they are based on. GTK+ optionally loads theGNOMEAccessibility Implementation Library (GAIL), an implementation of the accessibility interfaces defined by ATK, which bridges between the GNOME widgets and ATK. GAIL is provided by GTK in libgail-util. A second library is used in order to bridge between ATK and AT-SPI.

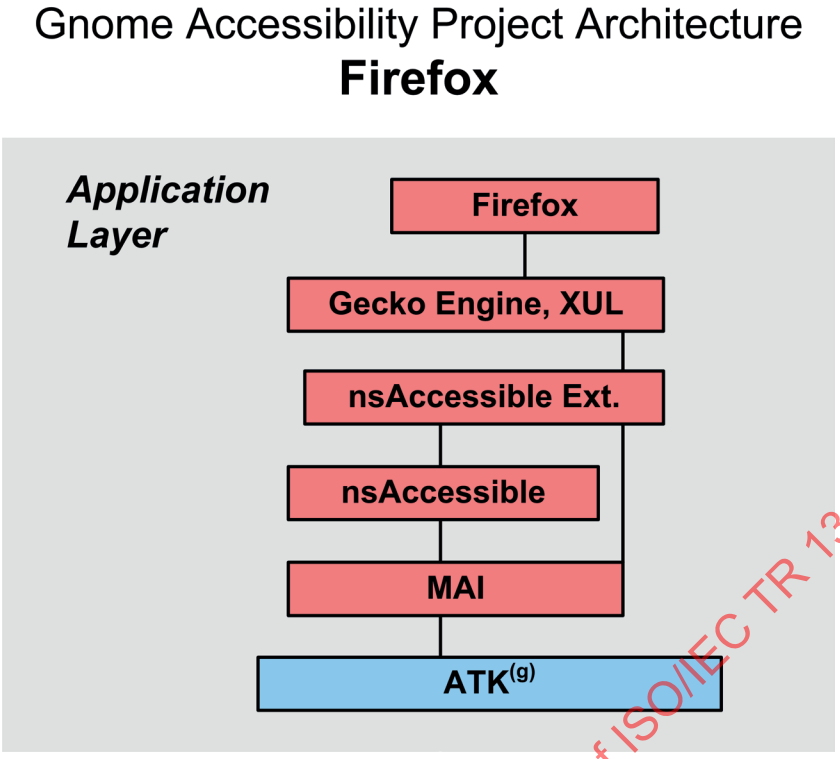


Key

(g) GNOME

Figure 3 — GNOME Mag and the GNOMEShell

— Mozilla-based applications like Firefox and Thunderbird do not use GTK+ but implement the ATK API directly within its applications using XUL Accessibility. The Mozilla Accessibility Architecture is described at https://developer.mozilla.org/en-US/docs/Web/Accessibility/Architecture and the XUL Accessibility Guidelines are published at https://developer.mozilla.org/en/XUL_accessibility_guidelines.

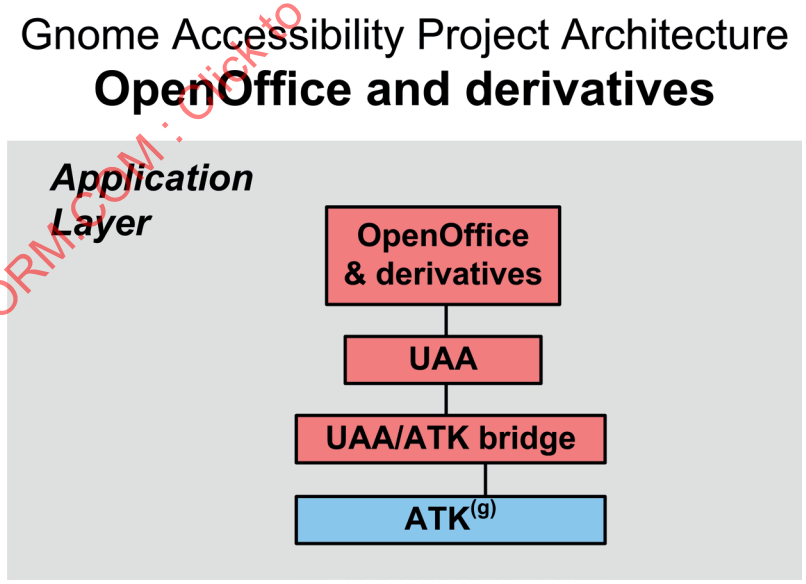


Key

(g) GNOME

Figure 4 — Mozilla Applications

— The Apache OpenOffice and LibreOffice applications bridge to ATK using the UNO Accessibility API, http://www.openoffice.org/ui/accessibility/unoapi.html.

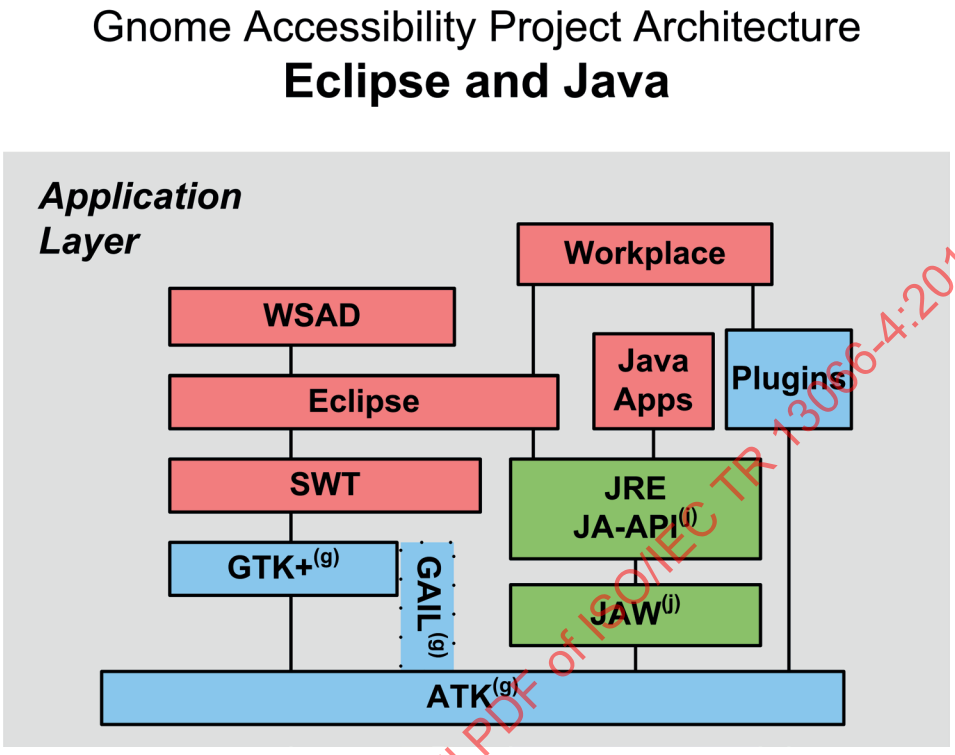


Key

(g) GNOME

Figure 5 — OpenOffice and derivatives

— Java applications may either use the Java accessibility framework which directly bridges to AT-SPI, or the new Java ATKwrapper https://git.gnome.org/browse/java-atk-wrapper/ Some Java applications, built using the Eclipse libraries, simply use GTK+ directly, as show in in Figure 6.



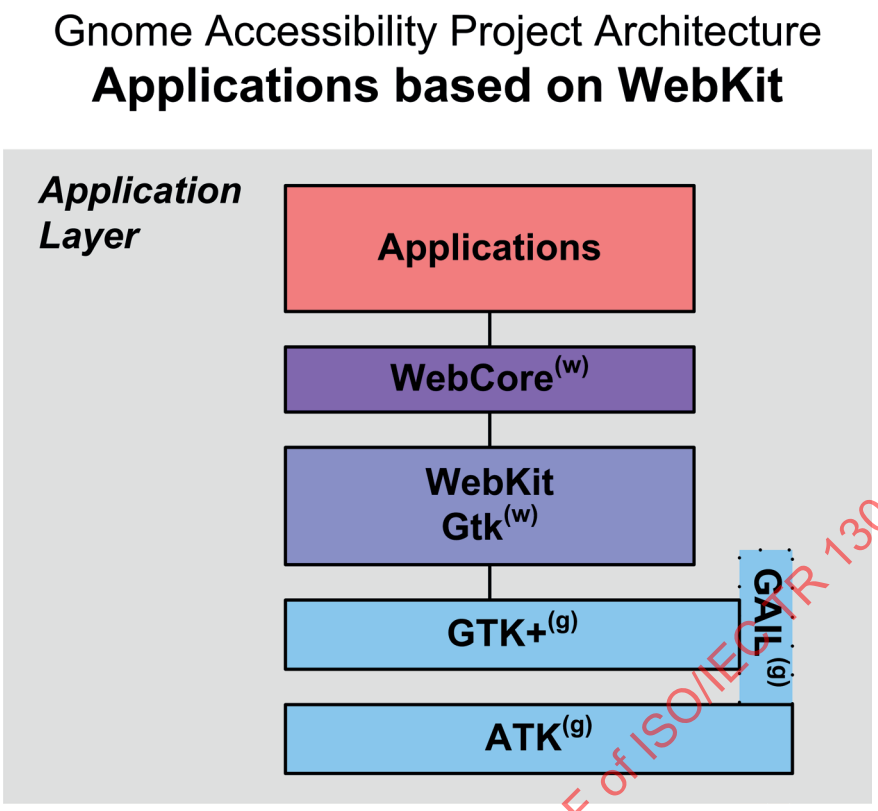
Key

(g) GNOME

(j) Java; everything else is Application

Figure 6 — Java and Eclipse Applications

— WebKit support is being provided directly through WebKitGTK+, http://webkitGTK.org.



Key

(g) GNOME

(w) WebKit; everything else is Application

Figure 7 — Applications based on WebKit

— KDE support utilizes a Qt plugin that bridges the QAccessible APIs to AT-SPI. The current approach is unique in that it does not bridge via ATK, but rather bridges directly to AT-SPI through QT-AT-SPI, http://gitorious.org/qt-at-spi.

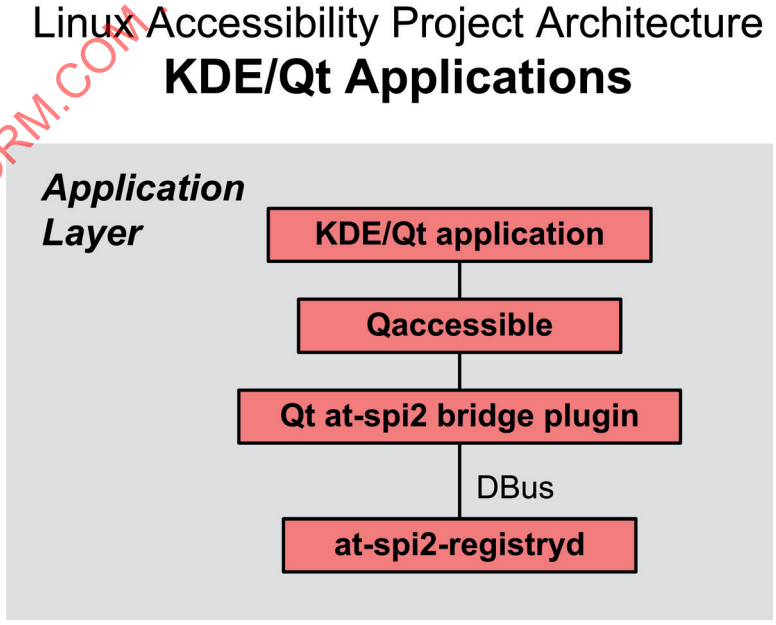


Figure 8 — Applications based on KDE/Qt

In the context of AT-SPI/ATK, applications are often called servers.

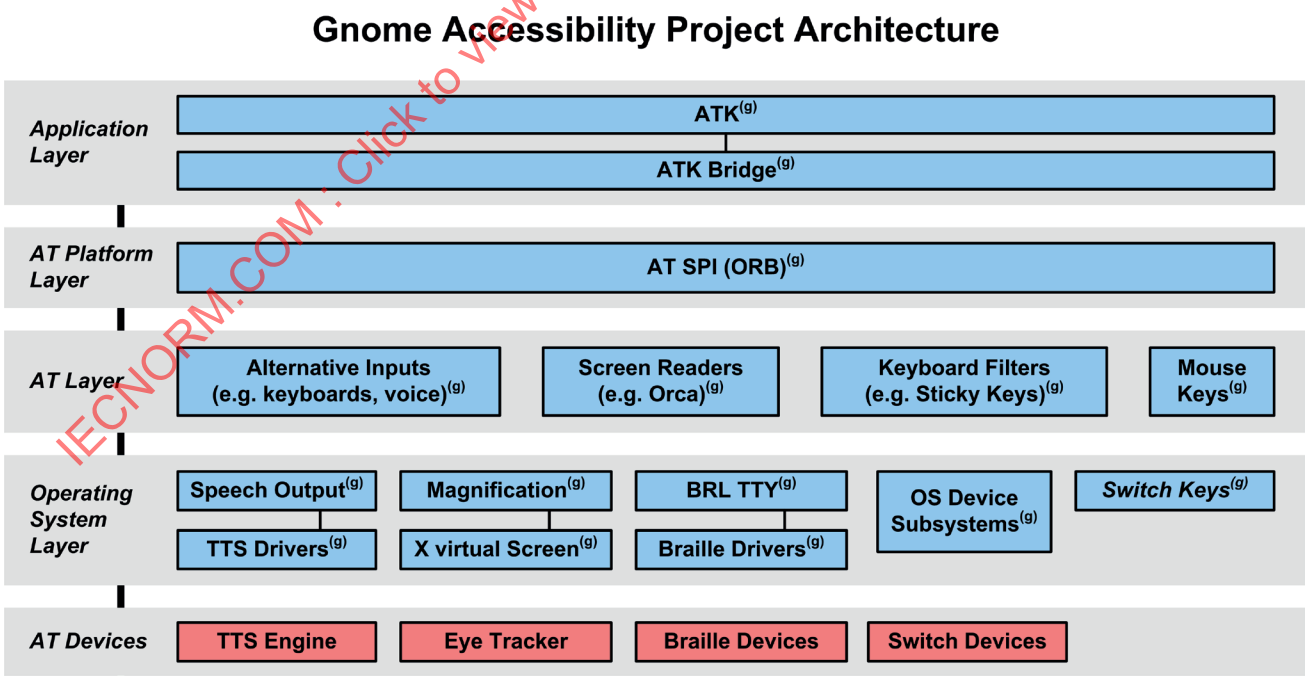
In order for an application to be accessible on the Linux/UNIX graphical desktop, it generally needs to provide information about its user interface using ATK. The sole exception to date is KDE where Qt applications are planned to bridge directly to AT-SPI.

If an application is written using GNOME's GTK+, all of the standard widgets provide the needed information to ATK and therefore the application will by default be accessible. Applications that do not use GTK+ (or UNO, XUL, Java, Qt, or Webkit) for their user interface, as well as any custom widgets, require additional work to make them accessible.

3.2.2 AT-SPI Aware Assistive Technologies

Assistive technologies are applications that are needed to obtain information about the user interfaces of applications. Screen readers are assistive technologies commonly utilized by computer users who are blind or severely visually impaired. A screen reader (like Orca, see https://live.gnome.org/Orca) needs to know what to present to the user through synthetic text-to-speech (TTS) or through a refreshable braille display. On-screen keyboards are assistive technologies commonly utilized by computer users who cannot use the standard keyboard (QUERTY or other layout) or a mouse. An on-screen keyboard (like Caribou, see https://live.gnome.org/Caribou) needs to send keyboard and mouse events to the application. In the context of AT-SPI, assistive technologies are often called clients because they consume and interact with application UI information.

The accessibility broker, ATK-Bridge (libatk-bridge), is a daemon that coordinates communication between ATK aware applications and assistive technologies. Also referred to as the accessibility bridge, this daemon necessarily links both to ATK and AT-SPI. Bridges are needed because different toolkits commonly implement similar features differently. Bridges to AT-SPI ensure that AT receives consistently formulated data. Each ATK aware application registers with the broker in order to offer its information. Assistive technologies may add event listeners to the broker, so that they get informed when accessibility related information in any application changes. They may also make queries of applications directly.



Key

(g) GNOME; everything else is Assistive technology

Figure 9 — ATK to AT layer

3.3 Support apart from AT-SPI/ATK

Persons unable to use a keyboard and mouse sometimes use alternative input technology supported through AT-SPI/ATK. However, many users can be accommodated programmatically through software that causes a standard keyboard to behave differently. Many of these features and behaviours have long been available in the XKeyboard Extension: Library Specification Library (X Version 11, Release 6.4) http://accessibility.linuxfoundation.org/a11yspecs/kbd/kafs.html#xkb-ref.

Individuals with mobility impairments benefit by having such features built-in and available through standard activation strategies, such as tapping the Shift key five times to activate StickyKeys. The routines provided by this API also benefit assistive technologies such as on screen keyboard and screen reader applications which do rely on AT-SPI/ATK.

The "Keyboard Access Functional Specification (KAFS)," http://accessibility.linuxfoundation.org/allyspecs/kbd/kafs.html, developed by the Open A11y Workgroup, identifies and adopts a subset of the XKBKeyboard Extension specification in order to provide standard keyboard features and behaviours required by persons with mobility impairments. A companion document, "Generic Test Assertions for Manual Testing (KAFS-GTA)," http://accessibility.linuxfoundation.org/a11yspecs/kbd/kafs-gta.html, which identifies how to test for compliance with KAFS, is also provided, http://accessibility.linuxfoundation.org/a11yspecs/kbd/kafs-gta.html.

KAFS defines the minimum level of keyboard accessibility support and notification that must be provided to the user by a Linux Foundation certified X Windowing System. This support ensures that users with a variety of physical disabilities will always have a basic level of access to the windowing system's core functions - keyboard input and controlling the system pointer (where available).

The features in KAFS are largely dependent on support provided at the operating system level of the platform. KAFS covers the 3 areas described below.

— "Configuration and Setting Requirements" enumerates the features and associated controls, configurability, and minimum ranges provided to the user.

— "End-User Notification, Keyboard Invocation, and Pointer Emulation Requirements" enumerates the additional notification and controllability a platform must support in the user interface developed to meet the Configuration and Settings Requirements.

— "Feature Behaviour Requirements" describes the type of behaviour that must occur when the functionality of a keyboard access feature is exercised.

4 Using the API

4.1 Overview

As in other environments, Linux/UNIX developers need to identify whether the role of their code is as a server, a provider of AT-SPI/ATK information, or a client, a consumer of AT-SPI/ATK information. Typically AT systems are AT-SPI clients, and query the AT-SPI/ATK 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, e.g. a screen reader is an AT-SPI client because it connects to the server and, using the AT-SPI/ATK APIs, interrogates the server for information about the object, such as its name and value or state, and presents information about the object to the user in a way that is meaningful. The AT-SPI/ATK server exposes accessibility information for component GUI controls in application software, such as focus changes.

4.2 User Interface elements

Like MSAA, IAccessible2, and Java, AT-SPI/ATK objects have roles and states. The AT-SPI/ATK interfaces define an extended set of accessible roles and states. In addition, an AT-SPI/ATK object will have properties describing its relationship to other objects, its position within a group of objects, its locale, etc.

AT-SPI/ATK provides special interfaces for certain types of accessible objects. The following interfaces expose properties that are unique to the particular type of object supported:

`Accessibility::Hypertext`

`AtkHypertext`

This interface is implemented for widgets that contain hypertext content. In most cases, this implies that the `Text` interface is also implemented. However, it is possible that this interface is implemented for a graphical image, in which case the `Text` interface is not implemented. The `Hypertext` interface contains methods for retrieving Hyperlink objects for hyperlinks within the text. However, this interface does not contain methods for retrieving text or for carat handling as these are handled within the `Text` interface.

`Accessibility::Hyperlink`

`AtkHyperlink`

This interface is `AtkHyperlink` is an ATK object which encapsulates a link or set of links (for instance, in the case of client-side image maps) in a hypertext document. It may implement the `AtkAction` interface. `AtkHyperlink` is particularly useful because it may also be used to refer to inline embedded content, since it allows specification of a start and end offset within the host `AtkHypertext` object.

The `AtkHyperlink` structure is not accessed directly. See https://library.gnome.org/devel/atk/stable/AtkHyperlink.html and https://developer.gnome.org/libatspi/stable/AtspiHyperlink.html for correct usage.

`Accessibility::Image`

`AtkImage`

This interface is implemented by components which expose image or pixmap content on screen. It contains methods for retrieving the screen position of the image and for retrieving or setting an alternative textual description of the image.

`Accessibility::Table`

`AtkTable`

This interface is implemented for UI components which contain tabular or row/column information. The `Table` interface is implemented for widgets that order their children, such as cells within a table. It contains methods for retrieving row and column headers, for retrieving a description of the contents of the table, and for translating between cell positions and child ID numbers.

`AtkTable` may also be used to present tree-structured information if the nodes of the trees can be said to contain multiple “columns.” Individual elements of an `AtkTable` are typically referred to as \cells,” and these cells are exposed by `AtkTable` as child `AtkObjects` of the `AtkTable`. Both row/column and child-index-based access to these children is provided.

See also 5.3.

4.3 Getting and setting focus

`Accessibility::Component`

`AtkComponent`

The `Component` interface is implemented for all widgets that have a screen representation. It contains methods for retrieving the screen position, adding and removing a focus handler, and grabbing the focus.

`AtkComponent` is implemented by most, if not all, UI elements with an actual on-screen presence, i.e. components which can be said to have a screen-coordinate bounding box. Virtually all widgets will

need to have `AtkComponent` implementations provided for their corresponding `AtkObject` class. In short, only UI elements which are not GUI elements will omit this ATK interface.

A possible exception might be textual information with a transparent background, in which case text glyph bounding box information is provided by `AtkText`.

4.4 Communication mechanisms

As has been noted above, IPC/RPC communication between applications and AT on Linux/UNIX graphical desktops is mediated by AT-SPI. As of this writing, these communications are achieved using Common Object Request Broker Architecture (CORBA), an inter-process communication standard defined by the Object Management Group that enables software components to communicate with each other across processes, address spaces, and even computers, in an object-to-object fashion. AT-SPI communication in CORBA is implemented by object request brokers (ORB). The CORBA specification dictates the presence of an ORB through which the application interacts with other objects. Thus, AT-SPI over CORBA defines a synchronous object to object communication mechanism.

Beginning with the release of GNOME 3.0 in calendar year 2011, CORBA has been deprecated by GNOME in favor of the D-Bus IPC/RPC technology, an inter-process communication technology developed by freedesktop.org that is the de facto standard of Linux/UNIXUNIX graphical environments.

D-Bus is message and bus-based. Its messages contain headers and metadata, as well as payload. D-Bus provides a daemon, known as the message bus daemon, that asynchronously routes messages between processes on a specific bus - the accessibility (A11Y) bus in the case of AT-SPI 2.0.

4.5 How GNOME uses the ATK/AT-SPI accessibility Application Programming Interface

As has been noted, accessibility can be supported on Linux/UNIX graphical desktop environments through several toolkits that already know how to talk to AT-SPI/ATK, - toolkits such as GTK, UNO, XUL, and Java/Swing. This part of ISO 13066 focuses on GTK as the more common and universal toolkit option.

The GTK+ accessibility architecture depends on a number of libraries and interfaces for programming a GTK+ application, and for making the objects and interfaces accessible, translatable, and usable.

a) The GLib library provides portability and convenience functions, generic data structures, and the GLib main loop.

b) The GDK library provides an abstraction layer between the GTK+ widgets in an application and the underlying X window system.

c) The GTK+ Object System provides a library and class hierarchy of objects (including widgets, containers, windows) with features that include inheritance, polymorphism and reference counting, signal notification, and attributes.

d) The Pango engine and library provide functions for text layout and rendering, with internationalization considerations.

e) The ATK library provides all the accessible object classes (based on GObject) and the interfaces the accessible objects implement (based on GInterface).

f) ATK is an in-process accessibility API, written to by applications and implemented for GTK+ widgets by the GAIL (GNOME Accessibility Implementation Library), which a GTK application dynamically loads at runtime if the value of the \accessibility GSettings key," /desktop/gnome/interface/accessibility is "true." In ATK/AT-SPI2 this key has been renamed to toolkit-accessibility (and is found in the master of gsettings-desktop-schemas).

g) AT-SPI is an out-of-process API implementation used by assistive technologies. The ATK bridge exports ATK to AT-SPI for applications written using the GTK+ toolkit, and for applications written using other toolkits that support AT-SPI/ATK.

5 Exposing User Interface Element Information

ISO/IEC 13066-1:2011, 7.1.7 requires that applications provide AT with information about user interface elements, including but not limited to the following:

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.

The following subclauses describe how AT-SPI/ATK supports each of the requirements in ISO/IEC 13066-1:2011, 7.1.7.

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

ATK (and AT-SPI) create a hierarchy of UI elements to present the UI to ATs. ATK has a defined set of roles that each UI element can use to describe the type of element and how to interact with it, e.g. a button on the screen would report that its role is ATK_ROLE_PUSH_BUTTON.

The AT-SPI model of an application is a tree where each node is an accessible object representing a UI element or sub-element.

— The root represents the application.

— An object node can have 0 or more children.

— An object may also have relations that specify “links” to other objects that are not direct children.

— Each object has a role that defines what it does, what states it can have and what operations can be performed on it through various methods.

— States represent object properties that may change over time such as `STATE_VISIBLE` or `STATE_CHECKED`.

— Attributes may be set as application-specific name-value pairs.

— Related methods and properties are grouped together into interfaces.

— An object only implements those interfaces that make sense for its role, e.g. an object that represents a piece of text may have a role of `ROLE_TEXT` and may implement the `Text` interface among others.

— The Accessible interface is implemented by all objects and provides basic accessibility information such as role, state, and children.

— The Component interface describes UI component properties such as position on screen.

— The Action interface allows the developer to programmatically invoke actions such as actions on an object of `ROLE_BUTTON` allowing the simulation of user operation through a “click” action.

`Accessibility::Role`

`Accessibility::State`

Both ATK and AT-SPI provide an accessibility object for each user-visible element in the user interface. The basic class for these objects, Accessible within AT-SPI and AtkObject within ATK, only contain methods for information that is common for all widgets, e.g. the state of the widget, relations to other widgets, references to child widgets, and a reference to the parent.

Please consult https://developer.gnome.org/libatspi/stable/ for the comprehensive enumerated list of roles and states available through AT-SPI.

`Accessibility::Accessible`

`AtkObject`

This is the base interface which is implemented by all accessible objects. All objects support interfaces for querying their contained "children" and position in the accessible-object hierarchy, whether or not they actually have children.

Data provided for the accessibility::accessible and `AtkObject` interfaces includes the following:

— the name of the widget;

— a description for the widget;

— the relation to other objects;

— the role of the widget within the user interface;

— the state of the widget;

— methods for handling notification when widget properties change;

— methods for determining the parent and children of the widget.

For a complete exposition, consult https://library.gnome.org/devel/atk/stable/AtkObject.html and https://developer.gnome.org/libatspi/stable/AtspiAccessible.html.

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

`Accessibility::Value`

`AtkValue`

The `Value` interface is implemented for widgets that represent a value within a bounded range of values. It contains methods for retrieving the minimum, maximum, and the current value and for changing the value. It is possible to set the Value interface to be read-only in which case any attempt to change the current value will fail.

For a complete exposition, consult https://library.gnome.org/devel/atk/stable/AtkValue.html and https://developer.gnome.org/libatspi/stable/libatspi-atspi-value.html.

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

`Accessibility::Document`

`AtkDocument`

This interface is implemented for widgets that are associated with the Document Object Model (DOM). It provides a mechanism for AT clients to access the DOM.

`Accessibility::Text`

`AtkText`

This interface is implemented by components with text content. The `text` interface is implemented by objects which place textual information onscreen as character strings or glyphs. The text interface allows access to the following:

— textual content, including display attributes, and semantic hints associated with runs of text;

— bounding box information for glyphs and substrings;

— it also allows portions of textual content to be selected, if the object's StateSet includes `STATE_SELECTABLE_TEXT`;

— provides not only traversal facilities and change notification for text content, but also caret tracking and glyph bounding box calculations.

This interface does not contain methods for changing text as this is handled within the `EditableText` interface.

`Accessibility::EditableText`

`AtkEditableText`

This interface is implemented for widgets that contain user-editable text content. This implies that the `Text` interface is also implemented. `EditableText` contains methods for the following:

— inserting a given text string at a given offset;

— deleting some part of the text;

— support for standard cut, copy, and paste functions.

This interface does not contain methods for retrieving text content, or for handling the carat as these functions are handled within the `Text` interface.

5.4 The location of the user interface element in relation to other user interface elements

`Accessibility::Relation`

`AtkRelation`

`Accessibility::RelationSet`

`AtkRelationSet`

`Accessibility::RelationType`

`AtkRelationType`

These interfaces provide comprehensive data regarding an object's relationship to other objects.

A common example of a relation is the "labelled by" and "label for" relations as expressed through `ATK_RELATION_LABELLED_BY and ATK_RELATION_LABEL_FOR`.

— `ATK_RELATION_LABELLED_BY` indicates an object is labelled by one or more target objects.

— `ATK_RELATION_LABEL_FOR` indicates an object is a label for one or more target objects.

For a complete exposition, consult the following:

— `AtkRelation` at https://library.gnome.org/devel/atk/stable/AtkRelation.html;

— `AtkRelation` at https://library.gnome.org/devel/atk/stable/AtkRelationSet.html;

— `AtkRelation` at https://library.gnome.org/devel/atk/stable/AtkRelationType.html;

— `Accessibility::Relation` at https://developer.gnome.org/libatspi/stable/AtspiRelation.html;

— `Accessibility::RelationSet` at https://people.gnome.org/~billh/at-spi-idl/html/namespaceAccessibility.html;

— `Accessibility::RelationType` at https://people.gnome.org/~billh/at-spi-idl/html/namespaceAccessibility.html.

As table structure relationships can be complex, AT-SPI/ATK provides a special interface AtkTable for specifying relationships among table cells. See 4.2.

6 Exposing User Interface Element Actions

ISO/IEC 13066-1:2011, 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.

`Accessibility::Action`

`AtkAction`

This interface is implemented by instances of `AtkObject` classes with which the user can interact directly, i.e. buttons, checkboxes, scrollbars, e.g. components which are not "passive" providers of UI information.

Exceptions: when the user interaction is already covered by another appropriate interface such as `AtkEditableText` (insert, delete, etc.), or `AtkValue` (set, value), then these actions are not also exposed by `AtkAction`.

Also, note that the `AtkAction` API is limited in that parameters may not be passed to the object being activated. Thus, the action must be self-contained and specifiable in a single "verb". Examples of appropriate use include: "press", "release", "click" for buttons, "drag" (meaning initiate drag) and "drop" for drag sources and drop targets, etc.

Because most UI interactions on components must be invocable via keyboard as well as mouse, there will generally be a close mapping between "mouse actions" that are possible on a component and the `AtkAction`. Where mouse and keyboard actions are redundant, `AtkAction` exposes only one action rather than exposing redundant actions, if possible.

For a complete exposition, see https://library.gnome.org/devel/atk/stable/AtkAction.html and https://people.gnome.org/~billh/at-spi-idl/html/classAccessibility_1_1Action.html.

7 Keyboard focus

ISO/IEC 13066-1:2011, 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.

