Adobe Acrobat 7.0.5

Reading PDF Through Accessibility Interfaces

November 15, 2005

Adobe Solutions Network — http://partners.adobe.com

Contents

Chapter 1 Accessibility for PDF 5

Overview of Accessibility for PDF 5

Rendering Order vs. Logical Order of Content 6

Document vs. Page Access 6

Inaccessible Documents 6

Protected Documents 7

Empty Documents 7

Unavailable Documents 8

Event Notifications 8

Retrieving an MSAA Object for an Event 9

Retrieving a PDF DOM Object for an Event 9

Chapter 2 Reading PDF Files Through MSAA 11

Overview of MSAA PDF Access 11

Retrieving IAccessible Objects 11

Acrobat Implementation of IAccessible Objects. 11

Identifying IAccessible Objects in a Document 12

IAccessible Method Summary 14

IAccessible Object Types for PDF 19

PDF Document. 20

PDF Page 21

PDF Protected Document 21

Empty PDF Document 22

PDF Structure Element. 23

PDF Content Element 25

PDF Comment 26

PDF Link 28

PDF Text Form Field 29

PDF Button Form Field. 29

PDF CheckBox Form Field 30

PDF RadioButton Form Field 31

PDF ComboBox Form Field 32

PDF List Box Form Field 34

PDF Digital Signature Form Field 36

PDF Caret 37

Chapter 3 Reading PDF Files Through the DOM Interface 39

IPDDomNode Data Types. 40

IPDDomNode Methods 42

Words and Lines in Text 45

IPDDomNodeExt Methods 46

IPDDomDocument Methods. 48

IPDDomElement Methods 50

Accessibility Attributes 51

IPDDomWord Methods 52

Chapter 4 Reading PDF Files on Adobe Reader for UNIX 53

Setting up Adobe Reader for Accessibility 54

Navigating the Hierarchy of Accessible Objects. 54

Accessible Object Interfaces 57

Action Interface 58

Component Interface 58

Text Interface 58

Accessing the User Interface. 59

Accessing PDF Content 60

1 Accessibility for PDF

Overview of Accessibility for PDF

The Adobe® Portable Document Format (PDF) is a file format for representing documents in a manner independent of the application software, hardware, and operating system used to create them, as well as of the output device on which they are to be displayed or printed. PDF files specify the appearance of pages in a document in a reliable, device independent manner.

Adobe provides methods to make the content of a PDF file available to assistive technology such as screen readers:

- On the Microsoft® Windows® operating system, Adobe Acrobat® and Adobe Reader® export PDF content as COM objects. Accessibility applications such as screen readers can interface with Acrobat or Adobe Reader in two ways:
  - Through Microsoft's Active Accessibility (MSAA) interface, using MSAA objects that Acrobat or Adobe Reader exports. See "Reading PDF Files Through MSAA" on page 11.
  - Directly through exported COM objects that allow access to the PDF document's internal structure, called the *document object model* (DOM). See "Reading PDF Files Through the DOM Interface" on page 39.

  The DOM and MSAA models are related, and developers can use either or both. Acrobat issues notifications to accessibility clients about interesting events occurring in the PDF file window and responds to requests from such clients.
- On Unix® platforms, Adobe Reader supports the Gnome accessibility architecture. C-based Accessibility Toolkit (ATK) interfaces are available. See "Reading PDF Files on Adobe Reader for UNIX" on page 53.

Recent versions of Acrobat have enhanced the support for accessibility interfaces:

- MSAA interfaces are supported in Acrobat 5.0 and higher.
- In Acrobat 6.0 and higher, information about the underlying PDF structure is made available through direct COM objects that represent the PDF DOM. The DOM accessibility interfaces provide somewhat more extensive access.
- In Acrobat 7.0, ATK and expanded DOM support is available.
  - The Linux®, Solaris™, AIX®, and HP-UX versions of Adobe Reader implement C-based ATK interfaces, allowing screen readers, screen magnifiers, and on-screen keyboards to query an Accessibility Technology - Service Providers Interface (AT-SPI) registry for applications that are accessible.
  - The DOM has been expanded to provide enhanced Caret, Selection, and Focus support (see "Event Notifications" on page 8), as well as the new interfaces `IPDDomDocument`, `ISelectText`, and `IPDDomNodeExt`.

This document is intended for developers of accessibility clients such as screen readers. It assumes the developer is familiar with COM, MSAA, or ATK, and the AcrobatAPI.

For information on Accessibility support in Acrobat, see:

http://partners.adobe.com/asn/tech/pdf/acrobatsdks.jsp

Information on MSAA can be found at

http://msdn.microsoft.com/library/en-us/msaa/msaastart_9w2t.asp

The rest of this chapter discusses issues that are common to both MSAA and DOM.

Rendering Order vs. Logical Order of Content

When rendering documents on the screen, Acrobat provides visual fidelity in a device-independent manner. However, the order in which Acrobat renders characters is not necessarily the same as the order in which they are to be read. Acrobat does not use standard system services that are used by assistive technology to capture content displayed on the screen.

Tagged PDF, introduced in PDF 1.4, defines a *logical structure* for the document that corresponds to the logical order of the content, regardless of the order in which the content is rendered. Acrobat uses the logical structure of a Tagged PDF document to determine word order. Through the accessibility interfaces, Acrobat can deliver the text of the PDF file as Unicode and can also make active elements such as links and form fields accessible.

NOTE: Acrobat can determine the logical structure of an untagged PDF file to some extent, but the results may be less satisfactory.

Document vs. Page Access

Through the accessibility interfaces, Acrobat can deliver contents of the entire PDF document contents or only the current page, regardless of what part of the document is visible on the screen:

- Delivering the entire document permits assistive technology to search the document for the next link or next instance of text.
- Delivering individual pages is necessary for very large documents that might exhaust the resources of the assistive technology.

The user controls the delivery method using the Reading Preferences.

Inaccessible Documents

A document can be *inaccessible* for one of the following reasons:

- It is protected by security settings (see "Protected Documents" below)
- It is or appears empty (see "Empty Documents" below).
- In addition, it may be temporarily unavailable (see "Unavailable Documents" on page 8).

The interfaces treat inaccessible documents as follows:

- Acrobat exports an MSAA object from the document, whose type indicates the reason for the inaccessibility.
- In Acrobat 6.0, inaccessible documents did not export any PDF DOM objects; attempts to retrieve PDF DOM objects from it failed without indicating the reason.
- In Acrobat 7.0, the DOM interface returns objects that represent the document, and DOM methods can be used to find out why the document is inaccessible.

Protected Documents

A document may have security settings that make it inaccessible. This can occur under the following conditions:

- It uses 40-bit RC4 encryption, and the author has forbidden copying text and graphics.
- It uses 128-bit RC4 encryption, and the author has forbidden making the contents accessible.

In either case, the user must contact the document author to provide a version that permits accessibility.

The following occurs when a document has security settings that make it inaccessible:

- Acrobat exports an MSAA **`IAccessible`** object warning of a possible error (see "PDF Protected Document" on page 21). This object has the role **`ROLE_SYSTEM_TEXT`** and the name "**`Alert: Protection Failure`**".
- When using the DOM interface in Acrobat 7, **`GetDocInfo`** returns the status **`DocState_Protected`**

You can become an Adobe Trusted Partner, and create Trusted Assistive Technology. Trusted Partners are developers of assistive products that respect the copy protection of encrypted PDF files, and can gain access to 40-bit encrypted files. For more information on becoming a Trusted Partner, see http://partners.adobe.com/asn/tech/pdf/acrobatsdks.jsp.

Empty Documents

A document can be inaccessible because it is empty, or it can appear empty because of the way the PDF was created. For instance, scanned images that have not been run through an optical character recognition (OCR) tool appear to be empty. Malformed structure trees can also make a document appear empty.

The following occurs when a document appears to be empty:

- Acrobat exports an MSAA **`IAccessible`** object warning of a possible error (see "Empty PDF Document" on page 22). This object has the role **`ROLE_SYSTEM_TEXT`** and the name "**`Alert: Empty document`**". If Acrobat is delivering information a page at a time, a genuinely empty page also generates this warning.

- When using the DOM in Acrobat 7, **`GetDocInfo`** returns the status **`DocState_Empty`**.

Unavailable Documents

When a document is unavailable, Acrobat returns similar objects from MSAA and DOM. A document may be unavailable for one of several reasons:

- If Acrobat is still preparing the document for access and the assistive technology attempts to read the document, the MSAA object name is "**`Alert: Document being processed`**".
- If Acrobat is waiting for a document on the web to download to the disk, the MSAA object name is "**`Alert: Document downloading`**".
- If the user cancels processing so that the document will never be available, the MSAA object name is "**`Alert: Document unavailable`**".

In all these cases, when using the DOM, the status returned in **`GetDocInfo`** is **`DocState_Unavailable`**.

Event Notifications

Each open document in Acrobat is associated with its own window handle. All **`WinNotifyEvent`** notifications for any part of the document use that window handle. For the PDF window:

- If **`childID == CHILDID_SELF`** (that is, 0), the event is for the entire document or page.
- If the **`childID`** parameter of the notification is non-zero, the event is for an object within the window, such as a form field, link, comment, or some part of the page content such a line or paragraph of text.

For Acrobat 7.0 and later, the following occurs:

- If the selection is set or changes, **`VALUECHANGE`** is notified, with the **`childID`** of the **`IAccessible`** object containing the beginning of the selection.
- If the selection is set, **`SELECTION`** is notified on the document (with a **`childID`** of **`0`**).
- If the selection is cleared, **`SELECTIONREMOVE`** is notified on the document.
- If the selection is extended, **`SELECTIONADD`** is notified, except when it is extended via keyboard commands (in that case **`SELECTIONREMOVE`** followed by **`SELECTION`** is notified).
- A **`LOCATIONCHANGE`** notification is issued when the caret moves. **`SHOW`** and **`HIDE`** notifications are issued when the caret is activated and deactivated.

Retrieving an MSAA Object for an Event

You can retrieve an **`IAccessible`** object from event notifications by using the MSAA function **`AccessibleObjectFromEvent`**. This object represents the document or an element within the document.

Some events always return an object of a particular type. For others, you must determine the type of the object from the role and specific **`childID`** (see "Identifying IAccessible Objects in a Document" on page 12). The meaning of the event can be different for different types of objects.

Acrobat posts the following **`WinEvent`** notifications:

- **`EVENT_OBJECT_FOCUS`**: The document window, a link, a comment, or a form field has received keyboard focus.

  `AccessibleObjectFromEvent` returns the appropriate **`IAccessible`** object, either for the document or page itself or for the link, comment, or form field. The **`childID`** parameter identifies the object.
- **`EVENT_OBJECT_LOCATIONCHANGE`** : The caret (text cursor) has moved. If the caret is in a text edit field containing keyboard focus, the value of the text field may also have changed.

  The **`idObjectType`** parameter for this event is **`objid_caret`**. **`AccessibleObjectFromEvent`** returns an **`IAccessible`** object for the caret
- **`EVENT_OBJECT_STATECHANGE`**:
  - If the **`childID`** parameter is **`CHILDID_SELF`**, the current document or page has changed its state by opening or closing a comment. The client should update its copy of the document content. Only the **`IAccessible`** object for the comment changes when this occurs.
  - If **`childID`** is non-zero, it is the UID of the **`IAccessible`** object for a form field, such as a checkbox or radio button, whose state has changed.
- **`EVENT_OBJECT_VALUECHANGE`**: Either the active document has changed or a new section of the document is now visible. Acrobat issues this notification if the user has taken an action that changes what is visible on the screen. For instance, the user may have followed a link or scrolled the document.
  - If the **`childID`** parameter is **`CHILDID_SELF`**, a new document or page has been opened or the current content has changed. The client should update its cached value of the document or page.
  - If the **`childID`** parameter is not **`CHILDID_SELF`**, it identifies the content on the page that to which the user has turned his attention. For instance, if a page has scrolled or Acrobat has followed a link to a new page, it identifies the first visible content on the page. The client may wish to update its internal state about where it is reading the document.

Retrieving a PDF DOM Object for an Event

To retrieve a DOM object, you can do one of the following:

- Call the MSAA library function **`AccessibleObjectFromEvent`** to get an **`IAccessible`** object (as described above). Then call that **`IAccessible`** object's **`get_PDDomNode`** method (see "IGetPDDomNode Interface" on page 12) to get the corresponding DOM object.
- Call the MSAA library function **`AccessibleObjectFromWindow`** on the window containing the document and pass **`OBJID_NATIVEOM`** as the second parameter. This returns the DOM object for the root of the document.

2 Reading PDF Files Through MSAA

Overview of MSAA PDF Access

Microsoft Active Accessibility defines the **`IAccessible`** interface to applications. This interface consists of a set of methods and properties that are defined in the MSAA documentation.

Acrobat implements and exports a set of **`IAccessible`** objects of different types to represent a document, its pages, and other elements of the document hierarchy.

This chapter contains two major reference sections:

- "IAccessible Method Summary" on page 14 provides a brief syntax description of the MSAA **`IAccessible`** methods and properties. More complete information can be found at http://msdn.microsoft.com/library/en-us/msaa/msaastart_9w2t.asp.
- "IAccessible Object Types for PDF" on page 19 describes the **`IAccessible`** objects exported by Acrobat.

Retrieving IAccessible Objects

An MSAA client can retrieve an **`IAccessible`** object for a UI element in the following four ways:

- Set a **`WinEvent`** hook, receive a notification, and call **`AccessibleObjectFromEvent`** to retrieve an **`IAccessible`** interface pointer for the UI element that generated the event. See "Event Notifications" on page 8 for details.
- Call **`AccessibleObjectFromWindow`** and pass the UI element's window handle. Each open document in Acrobat is associated with its own window handle.
- Call **`AccessibleObjectFromPoint`** and pass a screen location that lies within the UI element's bounding rectangle.
- Call an **`IAccessible`** method such as **`accNavigate`** or **`get_accParent`** to move to a different **`IAccessible`** object.

Acrobat Implementation of IAccessible Objects

Each type of **`IAccessible`** object has a different implementation of the standard methods:

- Links, tables, and form fields are explicitly identified through MSAA.
- Headers, paragraphs, and other elements of document structure are only represented implicitly.

NOTE: These elements are explicit in the DOM interface; see "Reading PDF Files Through the DOM Interface" on page 39.

For each document, Acrobat builds a tree of `IAccessible` objects representing the document and its internal structure. Because there is just one window handle associated with the document, Acrobat posts all event notifications to that window. In each notification, a `childID` identifies an `IAccessible` object for an element in the document. For example, when the user tabs to the next link, the `EVENT_OBJECT_FOCUS` notification includes a `childID` that is the UID of the link object. See "Event Notifications" on page 8.

The following interfaces are exported from the `IAccessible` object by Acrobat:

IGetPDDomNode Interface

This interface exports one function, `get_PDDomNode`, which returns a DOM object. The methods described in Chapter 3, "Reading PDF Files Through the DOM Interface" can then be used on this object.

Method	Syntax	Description
`get_PDDomNode`	`HRESULT get_PDDomNode( VARIANT varID, IPDDomNode **ppDispDoc);`	Returns a DOM object whose methods are described in Chapter 3. `varID` is the same as for the other MSAA methods (see "Descriptive Properties and Methods" on page 15).

ISelectText Interface

In Acrobat 7.0, the `ISelectText` interface is an interface exported by the `IAccessible` objects. It exports one function, `selectText`, that sets the text selection, but specifies the end location via `IAccessible` objects instead of DOM nodes. The `ISelectText` interface is available from the root `IAccessible` object:

Method	Syntax	Description
`selectText`	`LRESULT selectText( long startAccID, long startIndex, long endAccID, long endIndex);`	Sets the text selection. `startAccID` and `endAccID` are the `accID` identifiers for the starting and ending `IAccessible` elements, and `startIndex` and `endIndex` are zero-based indexes into the text of those `IAccessible` objects.

Identifying IAccessible Objects in a Document

You can identify the type of an `IAccessible` object by using the `get_accRole` method to get its Role attribute. However, you must also distinguish individual objects from others of the same type. You can do this by means of a unique identifier (UID) defined by Acrobat.

The **IAccessible** objects defined by Acrobat export a private interface, **IAccID**, defined in the file **IAccID.h**. It contains one function, **get_accID**, which returns an identifier that is unique within the open document or page. Use this UID to determine when two **IAccessible** objects refer to the same element in the document.

When a value-change notification or a focus notification has a non-zero **childID**, the value of **childID** is the UID of one of the objects on the page or document. Use the UID to uniquely identify the object that is the target of the notification.

get_accID Syntax

HRESULT get_accID(long *id);

Parameters

id	*(Filled by the method)* Returns the unique identifier of the **IAccessible** object. Must not be **NULL**.

Returns

Always returns **s_ok**.

Example

```
IAccID *pID;
long uid;
/* query for the IAccID interface */
RESULT hr = pObj->QueryInterface (IID_IAccID,
                         reinterpret_cast<void **>(&pID));
if (!FAILED(hr))
{
     pID->get_accID(&uid);
     pID->Release();
}
```

NOTE: If you obtained the **IAccessible** object via a call to **AccessibleObjectFrom*XXX***, it is not possible to query directly for this private interface. In that case, you must use the following alternate code:

```
IServiceProvider *sp = NULL;
hr = n->QueryInterface(IID_IServiceProvider, (LPVOID*)&sp);
if (SUCCEEDED(hr) && sp) {
     hr = sp->QueryService(SID_AccID, IID_IAccID, (LPVOID*)&pID);
     sp->Release();
}
```

IAccessible Method Summary

This section provides, for quick reference, a brief syntax summary of the **`IAccessible`** interface methods as defined by MSAA. For complete information, see the MSAA documentation at http://msdn.microsoft.com/library/en-us/msaa/msaastart_9w2t.asp.

All methods return **`HRESULT`**. The methods and properties are organized into the following groups:

Navigation and Hierarchy

`accNavigate (long` *`navDir`*`, VARIANT` *`varStart`*`, VARIANT*` *`pvarEnd`*`);`

Traverses to another UI element within a container and retrieves the object. All visual objects support this method.

`navDir` `[in]`	The direction to navigate, in spatial order or logical order. The spatial navigation constants are:
	`NAVDIR_UP`
	`NAVDIR_DOWN`
	`NAVDIR_RIGHT`
	`NAVDIR_LEFT`
	The logical navigation constants are:
	`NAVDIR_FIRSTCHILD`
	`NAVDIR_LASTCHILD`
	`NAVDIR_NEXT`
	`NAVDIR_PREVIOUS`
	NOTE: All **`accNavigate`** methods in PDF objects support the logical navigation directions. Only a few (PDF Structure Element, PDF ComboBox Form Field, and PDF ListBox Form Field) support the spatial navigation directions. Spatial navigation is only supported where it is explicitly noted.

`varStart` `[in]`	**`CHILDID_SELF`** to start navigation at the object itself, a child ID to start at one of the object's child elements.
`pvarEnd` `[out, retval]`	Returns a structure that contains information about the destination object. See MSAA documentation for details.

`get_accChild (VARIANT` *`varChildID`*`, IDispatch**` *`ppdispChild`*`);`

Retrieves an **`IDispatch`** interface pointer for the specified child, if one exists. All objects support this property.

`varChildID` `[in]`	The child ID for which to obtain a pointer. This can be a UID or the 1-based index of the child to retrieve.
`ppdispChild` `[out, retval]`	Returns the address of the child’s **`IDispatch`** interface.

`get_accChildCount (long* pcountChildren);`

Retrieves the number of children that belong to this object. All objects support this property.

`pcountChildren` `[out, retval]`	Returns the number of children. The children are accessible objects or child elements. If the object has no children, this value is zero.

`get_accParent (IDispatch** ppdispParent);`

Retrieves an **`IDispatch`** interface pointer for the parent of this object. All objects support this property.

`ppdispParent` `[out, retval]`	Returns the address of the parent's **`IDispatch`** interface.

Descriptive Properties and Methods

`accDoDefaultAction (VARIANT varID);`

Performs the object's default action. Not all objects have a default action.

`varID` `[in]`	**`CHILDID_SELF`** to perform the action for the object itself, a child ID to perform the action for one of the object's child elements.

`get_accDefaultAction(VARIANT varID, BSTR* pszDefaultAction);`

Retrieves a string that describes the object's default action. Not all objects have a default action.

`varID` `[in]`	**`CHILDID_SELF`** to get information for the object itself, a child ID to get information for one of the object's child elements.
`pszDefaultAction` `[out, retval]`	Returns a localized string that describes the default action for the object, or **`NULL`** if this object has no default action.

`get_accDescription (VARIANT varID, BSTR* pszDescription);`

Retrieves a string that describes the visual appearance of the object. Not all objects have a description.

`varID` `[in]`	**`CHILDID_SELF`** to get information for the object itself, a child ID to get information for one of the object's child elements.
`pszDescription` `[out, retval]`	Returns a localized string that describes the object, or **`NULL`** if this object has no description.

get_accName (VARIANT *varID*, BSTR* *pszName*);

Retrieves the name of the object. All objects have a name.

varID [in]	**CHILDID_SELF** to get information for the object itself, a child ID to get information for one of the object's child elements.
pszName [out, retval]	Returns a localized string that contains the name of the object.

get_accRole (VARIANT *varID*, VARIANT* *pvarRole*);

Retrieves the role of the object. All objects have a role.

varID [in]	**CHILDID_SELF** to get information for the object itself, a child ID to get information for one of the object's child elements.
pvarRole [out, retval]	Returns a structure that contain an object role constant in its **IVal** member.

get_accState (VARIANT *varID*, VARIANT* *pvarState*);

Retrieves the state of the object. All objects have a state.

varID [in]	**CHILDID_SELF** to get information for the object itself, a child ID to get information for one of the object's child elements.
pvarRole [out, retval]	Returns a structure that contain an object state constant in its **IVal** member.

get_accValue (VARIANT *varID*, BSTR* *pszValue*);

Retrieves the value of the object. Not all objects have a value.

varID [in]	**CHILDID_SELF** to get information for the object itself, a child ID to get information for one of the object's child elements.
pszValue [out, retval]	Returns a localized string that contains the current value of the object.

Selection and Focus

`accSelect (long flagsSelect, VARIANT varID);`

Modifies the selection or moves the keyboard focus of the object. All objects that support selection or receive the keyboard focus support this method.

`flagsSelect` `[in]`	Flags that control how the selection or focus operation is performed. A logical OR of these **`SELFLAG`** constants:
	`SELFLAG_NONE`
	`SELFLAG_TAKEFOCUS`
	`SELFLAG_TAKESELECTION`
	`SELFLAG_EXTENDSELECTION`
	`SELFLAG_ADDSELECTION`
	`SELFLAG_REMOVESELECTION`

`varID` `[in]`	**`CHILDID_SELF`** to select the object itself, a child ID to select one of the object's child elements.

`get_accFocus (VARIANT* pvarID);`

Retrieves the object that has the keyboard focus. All objects that receive the keyboard focus support this property.

`pvarID` `[out, retval]`	Returns the address of a **`VARIANT`** structure that contains information about the object that has the focus. See MSAA documentation for details.

`get_accSelection (VARIANT* pvarChildren);`

Retrieves the selected children of the object. All objects that support selection support this property.

`pvarChildren` `[out, retval]`	Returns the address of a **`VARIANT`** structure that contains information about the selected children. See MSAA documentation for details.

Spatial Mapping

`accLocation (long* pxLeft, long* pyTop, long* pcxWidth, long* pcyHeight, VARIANT varID );`

Retrieves the object's current screen location. All visual objects support this method.

`pxLeft, pxTop` `[out]`	Return the x and y screen coordinates of the upper-left boundary of the object’s location. (The origin is the upper left corner of the screen.)
`pxWidth, pxHeight` `[in]`	Return the object’s width and height in pixels.

varID **[in]**	**CHILDID_SELF** to get information for the object itself, a child ID to get information for one of the object's child elements.

accHitTest (long, long, VARIANT* pvarID);

Retrieves the object at a specific screen location. All visual objects support this method.

pxLeft, pxTop **[in]**	The x and y screen coordinates of the point to test. (The origin is the upper left corner of the screen.)
pvarID **[out, retval]**	Address of a **VARIANT** structure that identifies the object at the specified point. The information returned depends on the location of the specified point in relation to the object whose **accHitTest** method is being called. You can use this method to determine whether the object at that point is a child of the object for which the method is called. For details, see MSAA documentation. **NOTE:** For PDF objects, hit testing has been implemented in a very basic way; it does not identify the boundaries of the object itself with fine granularity, but reports whether or not the tested location is within the bounding box of an element or subtree.

IAccessible Object Types for PDF

This section describes the MSAA **`IAccessible`** object types that are defined to represent PDF documents and their elements. For each object, its methods are listed along with notes on how the implementation is specific to the object type.

NOTE: Methods that are not listed are not implemented for a given object type.

The objects are:

- "PDF Document" on page 20
- "PDF Page" on page 21
- "PDF Protected Document" on page 21
- "Empty PDF Document" on page 22
- "PDF Structure Element" on page 23
- "PDF Content Element" on page 25
- "PDF Comment" on page 26
- "PDF Link" on page 28
- "PDF Text Form Field" on page 29
- "PDF Button Form Field" on page 29
- "PDF CheckBox Form Field" on page 30
- "PDF RadioButton Form Field" on page 31
- "PDF ComboBox Form Field" on page 32
- "PDF List Box Form Field" on page 34
- "PDF Digital Signature Form Field" on page 36
- "PDF Caret" on page 37

The following are some general notes:

- Form fields generally correspond closely to standard user interface elements described in the MSAA SDK document. Their **`IAccessible`** objects attempt to match the behavior described in Appendix A, "Supported User Interface Elements," of that document. An exception is the PDF combo box, which has a much simpler structure.
- Form fields, links, and comments, as well as the document as a whole, can take keyboard focus. Subparts of the document (sections, paragraphs, and so on) cannot take focus.
- A document's contents may be only partially visible on the screen. The **`get_accLocation`** method for a given object returns the screen location of the visible part of the object only. You can use this method to determine which portions of the content are visible.

PDF Document

This type of **IAccessible** object represents the contents of an entire PDF document. The subtree of **IAccessible** objects beneath the PDF Document object reflects the logical structure of the document.

NOTE: Content that is not part of the logical structure, such as page headers and footers, is not presented through the MSAA interface.

Method	Implementation notes
accHitTest	Returns the object at a given location if the location is within the document's bounding box.
accLocation	Returns the screen coordinates of the visible part of the document.
accNavigate	Does not support spatial navigation (**NAVDIR_UP**, **NAVDIR_DOWN**, **NAVDIR_RIGHT**, **NAVDIR_LEFT**).
accSelect	For **SELFLAG_TAKEFOCUS**, the focus is set to the window containing the document and the document is positioned at the beginning. The other **SELFLAG** values are not supported.
get_accChild	Returns a child object.
get_accChildCount	Returns the number of child objects beneath this one.
get_accDescription	The description contains the full path name of the document and the number of pages it contains: "fileName, XXX pages".
get_accFocus	Returns the object that has the keyboard focus if it is this object or its child.
get_accParent	The parent is **NULL**.
get_accRole	The role is **ROLE_SYSTEM_DOCUMENT**.
get_accSelection	Returns **NULL**.
get_accState	The state is **STATE_SYSTEM_READONLY**.
get_accValue	If the root of the structure tree has an **Alt** attribute, the value is the contents of the **Alt** attribute.

PDF Page

This type of **`IAccessible`** object represents the contents of one page of a PDF document. The subtree of **`IAccessible`** objects beneath the PDF Page node reflects the logical structure of the page.

NOTE: Content that is not part of the logical structure, such as page headers and footers, is not presented through the MSAA interface.

Method	Implementation notes
`accHitTest`	Returns the object at the given location if the location is within the page's bounding box.
`accLocation`	Returns the screen coordinates of the visible part of the page.
`accNavigate`	Does not support spatial navigation (**`NAVDIR_UP`**, **`NAVDIR_DOWN`**, **`NAVDIR_RIGHT`**, **`NAVDIR_LEFT`**).
`accSelect`	For **`SELFLAG_TAKEFOCUS`**, the focus is set to the window containing the page and the page is positioned at the top. The other **`SELFLAG`** values are not supported.
`get_accChild`	Returns a child object.
`get_accChildCount`	Returns the number of child objects beneath this one.
`get_accDescription`	The description contains the full path name of the document and the page number of the page: "fileName, page XXX".
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accParent`	The parent is **`NULL`**.
`get_accRole`	A custom role, **`Page`**, is defined for this object.
`get_accSelection`	Returns **`NULL`**.
`get_accState`	The state is **`STATE_SYSTEM_READONLY`**.
`get_accValue`	If the root of the structure tree has an **`Alt`** attribute, the value is the contents of the **`Alt`** attribute

PDF Protected Document

This type of **`IAccessible`** object represents a protected document. When the permissions associated with a document disable accessibility, the contents are not exported through the MSAA interface. The **`IAccessible`** object for such a document informs the client that the document is protected.

Method	Implementation notes
`accHitTest`	Returns **`NULL`**.
`accLocation`	The screen coordinates of the visible part of the document.
`accNavigate`	Does not support spatial navigation (**`NAVDIR_UP`**, **`NAVDIR_DOWN`**, **`NAVDIR_RIGHT`**, **`NAVDIR_LEFT`**).
`accSelect`	Returns **`NULL`**.
`get_accChildCount`	The child count is 0.
`get_accFocus`	Returns **`NULL`**.
`get_accName`	The name is "Alert: Protection Failure".
`get_accParent`	The parent is **`NULL`**.
`get_accRole`	The role is **`ROLE_SYSTEM_TEXT`**.
`get_accSelection`	Returns **`NULL`**.
`get_accState`	The state is **`STATE_SYSTEM_ALERT_MEDIUM + STATE_SYSTEM_UNAVAILABLE + STATE_SYSTEM_READONLY`**.
`get_accValue`	The value is "This documen's security settings prevent access."

Empty PDF Document

This type of **`IAccessible`** object represents an empty or apparently empty document. A PDF file may have no contents to export through MSAA if, for instance, the file is a scanned image that has not been run through an optical character recognition (OCR) tool. The **`IAccessible`** object for empty documents and pages informs the client that there may be a problem, even if the document or page is genuinely empty.

Method	Implementation notes
`accHitTest`	Returns **`NULL`**.
`accLocation`	Returns the screen coordinates of the visible part of the document.

Method	Implementation notes
`accNavigate`	Does not support spatial navigation (**`NAVDIR_UP`**, **`NAVDIR_DOWN`**, **`NAVDIR_RIGHT`**, **`NAVDIR_LEFT`**).
`accSelect`	Returns **`NULL`**.
`get_accChildCount`	The child count is 0.
`get_accFocus`	Returns **`NULL`**.
`get_accName`	The name is "Alert: Empty document".
`get_accParent`	The parent is **`NULL`**.
`get_accRole`	The role is **`ROLE_SYSTEM_TEXT`**.
`get_accSelection`	Returns **`NULL`**.
`get_accState`	The state is **`STATE_SYSTEM_READONLY`**.
`get_accValue`	The value is "This document appears to be empty. It may be a scanned image that needs OCR or it may have malformed structure."

PDF Structure Element

This type of **`IAccessible`** object represents a subtree of the logical structure tree for the document. It might correspond to a paragraph, a heading, a chapter, a span of text within a word, or a figure.

Method	Implementation notes
`accDoDefaultAction`	If the element has state **`STATE_SYSTEM_LINKED`**, performs the action associated with the link.
`accHitTest`	Returns this object or any child at the given location if the location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the visible part of the subtree.
`accNavigate`	Only spatial navigation (**`NAVDIR_UP`**, **`NAVDIR_DOWN`**, **`NAVDIR_RIGHT`**, **`NAVDIR_LEFT`**) is supported for table elements (**`ROLE_SYSTEM_CELL`**, **`ROLE_SYSTEM_ROW`**, **`ROLE_SYSTEM_ROWHEADER`**, **`ROW_SYSTEM_COLUMNHEADER`**).

Method	Implementation notes
`accSelect`	For `SELFLAG_TAKEFOCUS`, sets focus to the document window and positions the document to the beginning of the structure element content. The other `SELFLAG` values are not supported.
`get_accChild`	Returns a child object.
`get_accChildCount`	Returns the number of child objects beneath this one. If the node has an **Alt** or **ActualText** attribute, the child count is always zero.
`get_accDefaultAction`	If the element has state `STATE_SYSTEM_LINKED`, returns a text description of the action associated with the link (such as "go to page 5" or "play movie").
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accParent`	The parent is either another structure element or the document structure root.

`get_accRole`	The role is one of:
	`ROLE_SYSTEM_GROUPING`
	`ROLE_SYSTEM_TABLE`
	`ROLE_SYSTEM_CELL`
	`ROLE_SYSTEM_ROW`
	`ROLE_SYSTEM_ROWHEADER`
	`ROW_SYSTEM_COLUMNHEADER`

`get_accSelection`	Returns `NULL`.

`get_accState`	The state is a logical OR of one or more of the following:
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_LINKED`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`
	● `STATE_SYSTEM_READONLY` is always set.
	● If the element is part of a link (that is, if it has an ancestor of role `ROLE_SYSTEM_LINK`) then both `STATE_SYSTEM_LINKED` and `STATE_SYSTEM_FOCUSABLE` are set, and `STATE_SYSTEM_FOCUSED` can also be set.

`get_accValue`	If this node has an `Alt` or `ActualText` attribute, the value is the contents of the attribute.

PDF Content Element

This type of `IAccessible` object corresponds to a leaf node of the logical structure tree for the document. It corresponds to marking commands in the page content stream.

Method	Implementation notes
`accDoDefaultAction`	If the element has state `STATE_SYSTEM_LINKED`, performs the action associated with the link.
`accHitTest`	Returns this object if the given location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the visible part of the element.
`accNavigate`	Does not support spatial navigation (`NAVDIR_UP`, `NAVDIR_DOWN`, `NAVDIR_RIGHT`, `NAVDIR_LEFT`).
`accSelect`	For `SELFLAG_TAKEFOCUS`, sets focus to the document window and positions the document to the beginning of the content. The other `SELFLAG` values are not supported.
`get_accChildCount`	The child count is 0.
`get_accDefaultAction`	If the element has state `STATE_SYSTEM_LINKED`, describes the action associated with the link.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accParent`	The parent is either a structure element or the document structure root.

`get_accRole`	The role is one of:
	`ROLE_SYSTEM_TEXT`
	`ROLE_SYSTEM_GRAPHIC`
	`ROLE_SYSTEM_CLIENT`

`get_accSelection`	Returns `NULL`.

Method	Implementation notes

`get_accState`	The state is a logical OR of one or more of the following:
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_LINKED`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`
	● `STATE_SYSTEM_READONLY` is always set.
	● If the element is part of a link (that is, if it has an ancestor of role `ROLE_SYSTEM_LINK`) then both `STATE_SYSTEM_LINKED` and `STATE_SYSTEM_FOCUSABLE` are set, and `STATE_SYSTEM_FOCUSED` can also be set.

`get_accValue`	If this node has an `Alt` or `ActualText` attribute, the value is the contents of that attribute. Otherwise, the value is all of the text contained in the marking commands for this node.

PDF Comment

This type of `IAccessible` object corresponds to a comment, such as a text note or highlight comment, attached to the document.

NOTE: PDF comments cover a range of objects, many of which do not map into the standard MSAA roles. The `IAccessible` object captures the most important properties of comments.

Method	Implementation notes
`accDoDefaultAction`	The default action depends on the type of comment. It can, for example, open or close a popup.
`accHitTest`	Returns this object if the given location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the visible part of the object.
`accNavigate`	Does not support spatial navigation (`NAVDIR_UP`, `NAVDIR_DOWN`, `NAVDIR_RIGHT`, `NAVDIR_LEFT`).
`accSelect`	Supports `SELFLAG_TAKEFOCUS` (that is, selecting the comment gives it the keyboard focus).
`get_accChildCount`	The child count is 0.

Method	Implementation notes
`get_accDefaultAction`	Describes the default action, which depends on the type of comment.
`get_accDescription`	For file attachment and sound comments, a description of the icon for the comment.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	● The name indicates the type of comment; for example, Text Comment or Underline Comment. ● If the comment is open and has a title, the name also contains the title of the comment. ● If the comment is a Free Text comment or modifies a span of text (such as an Underline or Strikeout Comment), the name also contains the text.
`get_accParent`	The parent is either a structure element or the document structure root.

`get_accRole`	The role is one of:
	`ROLE_SYSTEM_TEXT`
	`ROLE_SYSTEM_WHITESPACE`
	`ROLE_SYSTEM_PUSHBUTTON`

`get_accSelection`	Returns `NULL`.

`get_accState`	The state is a logical OR of one or more of the following:
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_LINKED`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_EXPANDED`
	`STATE_SYSTEM_COLLAPSED`
	`STATE_SYSTEM_FOCUSED`
	● If a comment can be opened, `STATE_SYSTEM_LINKED` is set.
	● `STATE_SYSTEM_EXPANDED` and `STATE_SYSTEM_COLLAPSED` indicate whether the comment is open.

`get_accValue`	● If the comment is open, the value is the contents of the comment pop-up window. ● If the comment is a type that does not open, the value is the contents of the comment itself.

PDF Link

This type of **`IAccessible`** object corresponds to a link in the document.

Method	Implementation notes
`accDoDefaultAction`	Performs the link's action.
`accHitTest`	Returns this object or any child at the given location if the location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the visible part of the object.
`accNavigate`	Does not support spatial navigation (**`NAVDIR_UP`**, **`NAVDIR_DOWN`**, **`NAVDIR_RIGHT`**, **`NAVDIR_LEFT`**).
`accSelect`	Supports **`SELFLAG_TAKEFOCUS`**
`get_accChild`	Returns a child object.
`get_accChildCount`	Returns the number of children. If the node has an **`Alt`** or **`ActualText`** attribute, the child count is always zero.
`get_accDefaultAction`	Describes the action defined for this link.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	If there is an **`Alt`** or **`ActualText`** attribute associated with this link, the name is the associated **`Alt`** text or **`ActualText`**. Otherwise,the name is the value of the first content child.
`get_accParent`	The parent is either a structure element or the document structure root.
`get_accRole`	The role is **`ROLE_SYSTEM_LINK`**.
`get_accSelection`	Returns **`NULL`**.

`get_accState`	The state is a logical OR of the following:
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_LINKED`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`

`get_accValue`	The value is a unique identifier for each link.

PDF Text Form Field

This type of **`IAccessible`** object corresponds to a text form field in the document.

Method	Implementation notes
`accDoDefaultAction`	Sets focus to the text field for editing.
`accHitTest`	Returns this object if the given location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the visible part of the object.
`accNavigate`	Does not support spatial navigation (**`NAVDIR_UP`**, **`NAVDIR_DOWN`**, **`NAVDIR_RIGHT`**, **`NAVDIR_LEFT`**).
`accSelect`	Supports **`SELFLAG_TAKEFOCUS`** (that is, selecting the field gives it the keyboard focus).
`get_accChildCount`	The child count is 0.
`get_accDefaultAction`	The default action is "DoubleClick", which sets the keyboard focus to this field.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	The user name (short description) of the form field.
`get_accParent`	Returns the parent object.
`get_accRole`	The role is **`ROLE_SYSTEM_TEXT`**.

`get_accState`	The state of the text field is a logical OR of one of more of:
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_UNAVAILABLE`
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_SELECTABLE`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`
	`STATE_SYSTEM_PROTECTED`

`get_accValue`	The value is the text in the text field.

PDF Button Form Field

This type of **`IAccessible`** object corresponds to a button form field in the document.

Method	Implementation notes
`accDoDefaultAction`	Presses the button.
`accHitTest`	Returns this object if the given location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the visible part of the object.
`accNavigate`	Does not support spatial navigation (`NAVDIR_UP`, `NAVDIR_DOWN`, `NAVDIR_RIGHT`, `NAVDIR_LEFT`).
`accSelect`	Supports `SELFLAG_TAKEFOCUS` (that is, selecting the field gives it the keyboard focus).
`get_accChildCount`	The child count is 0.
`get_accDefaultAction`	The default action is "Press".
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	The user name of the form field (short description).
`get_accParent`	Returns the parent object.
`get_accRole`	The role is `ROLE_SYSTEM_PUSHBUTTON`.

`get_accState`	The state of the button is a logical OR of one or more of:
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_UNAVAILABLE`
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`

PDF CheckBox Form Field

This type of `IAccessible` object corresponds to a checkbox form field in the document.

Method	Implementation notes
`accDoDefaultAction`	Checks or unchecks the box.
`accHitTest`	Returns this object if the given location is within the bounding box of this object.

Method	Implementation notes
`accLocation`	Returns the screen coordinates of the visible part of the object.
`accNavigate`	Does not support spatial navigation (`NAVDIR_UP`, `NAVDIR_DOWN`, `NAVDIR_RIGHT`, `NAVDIR_LEFT`).
`accSelect`	Supports `SELFLAG_TAKEFOCUS` (that is, selecting the field gives it the keyboard focus).
`get_accChildCount`	The child count is 0.
`get_accDefaultAction`	● If the check box has been selected, the default action is “UnCheck”. ● If the check box has not been selected, the default action is “Check”.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	The user name (shoft description) of the form field.
`get_accParent`	Returns the parent object.
`get_accRole`	The role is `ROLE_SYSTEM_CHECKBUTTON`.

`get_accState`	The state of the check box is a logical OR of one or more of:
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_UNAVAILABLE`
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`
	`STATE_SYSTEM_CHECKED`

PDF RadioButton Form Field

This type of `IAccessible` object corresponds to a radio button form field in the document.

Method	Implementation notes
`accDoDefaultAction`	Clicks the radio button.
`accHitTest`	Returns this object if the given location is within the bounding box of this object.

Method	Implementation notes
`accLocation`	Returns the screen coordinates of the visible part of the object.
`accNavigate`	Does not support spatial navigation (`NAVDIR_UP`, `NAVDIR_DOWN`, `NAVDIR_RIGHT`, `NAVDIR_LEFT`).
`accSelect`	Supports `SELFLAG_TAKEFOCUS` (that is, selecting the field gives it the keyboard focus).
`get_accChildCount`	The child count is 0.
`get_accDefaultAction`	The default action is “Check”.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	The user name (short description) of the form field.
`get_accParent`	Returns the parent object.
`get_accRole`	The role is `ROLE_SYSTEM_RADIOBUTTON`.

`get_accState`	The state of the radio button is a logical OR of one or more of:
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_UNAVAILABLE`
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`
	`STATE_SYSTEM_CHECKED`

PDF ComboBox Form Field

This type of `IAccessible` object corresponds to a combobox form field in the document. It can represent either the combo box itself, or a list item in a combobox.

Method	Implementation notes
`accDoDefaultAction`	● The combobox does not have a default action. ● For a list item, the default action is “DoubleClick”, which selects the list item.

Method	Implementation notes
`accHitTest`	• For a combo box, returns this object or any child at the given location if the location is within the bounding box of this object. • For a list item, returns this object if the given location is within the bounding box of this object.
`accLocation`	• For a combo box, returns the screen coordinates of the visible part of the object. • For a list item, the location is always reported as 0,0,0,0.
`accNavigate`	Spatial directions **`NAVDIR_UP`** and **`NAVDIR_DOWN`** are available for list items.
`accSelect`	• The combo box supports **`SELFLAG_TAKEFOCUS`** (that is, selecting the field gives it the keyboard focus). • For a list item, sets the combo box to the list item value.
`get_accChild`	• For a combo box, gets the child items. • A list item has no children.
`get_accChildCount`	• For a combo box, the child count is the number of items in the list. • For a list item, the child count is 0.
`get_accDefaultAction`	• The combobox does not have a default action. • For a list item, the default action is "DoubleClick", which selects the list item.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	• For a combo box, the name is the user name (short description) of the form field if it has been defined. • For a list item, the name is the text of the list item.
`get_accParent`	Returns the parent object.
`get_accSelection`	Returns **`NULL`**.
`get_accRole`	• For a combo box , the role is **`ROLE_SYSTEM_COMBOBOX`**. • For a list item, the role is **`ROLE_SYSTEM_LISTITEM`**

Method	Implementation notes

`get_accState`	● For a combo box , the state is a logical OR of one or more these values:
	`STATE_SYSTEM_INVISIBLEc`
	`STATE_SYSTEM_UNAVAILABLE`
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`
	`STATE_SYSTEM_SELECTABLE`
	`STATE_SYSTEM_SELECTED`
	● For a list box item, the state is a logical OR of one or more these values:
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_SELECTABLE`
	`STATE_SYSTEM_SELECTED`
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_UNAVAILABLE`

`get_accValue`	● For a combo box, the value is the text value of the currently selected list item. ● For a list item, the value is the text of the list item.

PDF List Box Form Field

This type of **`IAccessible`** object corresponds to a list box form field in the document. It can represent either the list box itself or a list item in a list box.

Method	Implementation notes
`accDoDefaultAction`	● The list box does not have a default action. ● For a list item, the default action is "Double Click," which selects the item.
`accHitTest`	● For a list box, returns this object or any child at the given location if the location is within the bounding box of this object. ● For a list item, returns this object if the given location is within the bounding box of this object.
`accLocation`	● For a list box, returns the screen coordinates of the visible part of the object. ● For a list item, the location is always reported as 0,0,0,0.

Method	Implementation notes
`accNavigate`	Spatial directions **`NAVDIR_UP`** and **`NAVDIR_DOWN`** are available for list items.
`accSelect`	● The list box supports **`SELFLAG_TAKEFOCUS`** (that is, selecting the field gives it the keyboard focus). ● For a list item, sets the list box selection to the list item value.
`get_accChild`	● For a list box, gets the child items. ● A list item has no children.
`get_accChildCount`	● For a list box, the child count is the number of items in the list box. ● For a list item, the child count is 0.
`get_accDefaultAction`	● The list box does not have a default action. ● For a list item, the default action is "Double Click," which selects the item.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	● For a list box, the name is the user name (short description) for the form field. ● For a list item, the name is the text of the list item.
`get_accParent`	Returns the parent object.
`get_accRole`	● For a list box , the role is **`ROLE_SYSTEM_LIST`**. ● For a list item, the role is **`ROLE_SYSTEM_LISTITEM`**

`get_accState`	● For a list box , the state is a logical OR of one or more these values:
	`STATE_SYSTEM_INVISIBLEc`
	`STATE_SYSTEM_UNAVAILABLE`
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_FOCUSABLE`
	● For a list item, the state is a logical OR of one or more these values:
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_SELECTABLE`
	`STATE_SYSTEM_SELECTED`
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_UNAVAILABLE`

`get_accSelection`	Returns **`NULL`**.

Method	Implementation notes
`get_accValue`	● For a list box , the value is the text value of the currently selected list item. ● For a list item, the **`Value`** attribute is the text of the list item.

PDF Digital Signature Form Field

This type of **`IAccessible`** object corresponds to a digital signature form field in the document.

Method	Implementation notes
`accDoDefaultAction`	Signs the document if the signature field is unsigned and has either been opened with Acrobat or the document has permissions that allow signing. If the document is signed, the default action brings up a dialog box containing the signature information.
`accHitTest`	Returns this object if the given location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the visible part of the object.
`accNavigate`	Does not support spatial navigation (**`NAVDIR_UP`**, **`NAVDIR_DOWN`**, **`NAVDIR_RIGHT`**, **`NAVDIR_LEFT`**).
`accSelect`	Supports **`SELFLAG_TAKEFOCUS`**.
`get_accChildCount`	The child count is 0.
`get_accDefaultAction`	Returns **`NULL`**.
`get_accFocus`	Returns the object that has the keyboard focus if it is this object or its child.
`get_accName`	The user name (short description) of the form field.
`get_accParent`	Returns the parent object.
`get_accRole`	The Digital Signature form field does not map to any of the existing roles, and a custom role, **`Signature`**, has been defined for it.

Method	Implementation notes

`get_accState`	The **`State`** attribute of the digital signature is a logical OR of one of more of these values:
	`STATE_SYSTEM_INVISIBLE`
	`STATE_SYSTEM_UNAVAILABLE`
	`STATE_SYSTEM_READONLY`
	`STATE_SYSTEM_FOCUSABLE`
	`STATE_SYSTEM_FOCUSED`
	`STATE_SYSTEM_CHECKED`
	`STATE_SYSTEM_TRAVERSED`
	● If **`STATE_SYSTEM_CHECKED`** is set, but not **`STATE_SYSTEM_TRAVERSED`**, the signature is unverified.
	● If **`STATE_SYSTEM_TRAVERSED`** is set, but not **`STATE_SYSTEM_CHECKED`**, the signature is invalid.
	● If both **`STATE_SYSTEM_CHECKED`** and **`STATE_SYSTEM_TRAVERSED`** are set, the signature is valid.

`get_accValue`	The **`Value`** attribute is the name and date of the signature, if that information is present.

PDF Caret

This type of **`IAccessible`** object represents a caret (text cursor). If a document contains the system caret because focus is within an editable text field or an editable ComboBox field, clients can obtain an **`IAccessible`** object for the caret to determine where is is located.

Method	Implementation notes
`accHitTest`	Returns this object if the given location is within the bounding box of this object.
`accLocation`	Returns the screen coordinates of the caret, both when the caret is in a form field and when it is in the document.
`get_accChildCount`	The child count is 0.
`get_accDescription`	The description is a string containing the index of the character in the field that follows the caret. If the caret is at the beginning of the field, the description string is “0”. If the caret follows the first character, the description string is “1”.

Method	Implementation notes
`get_accParent`	The parent is the field containing the caret. However, the caret `IAccessible` object is not listed among the children of that field's `IAccessible` object.
`get_accRole`	The role is `ROLE_SYSTEM_CARET`.
`get_accState`	The state is 0.
`get_accValue`	The value is the current value of the Text field or ComboBox form field containing the caret.

3 Reading PDF Files Through the DOM Interface

Acrobat 6.0 and higher defines a document object model (DOM) that provides more complete access to the document structure than the MSAA interface. The Accessibility plug-in defines and exports five COM interfaces in `AcrobatAccess.lib` that expose Acrobat's document hierarchy:

- `IPDDomNode` defines methods that apply to all elements of the document hierarchy (see "IPDDomNode Data Types" on page 40 and "IPDDomNode Methods" on page 42).
- `IPDDomDocument` interface is exported by the root object for the page or document (see "IPDDomDocument Methods" on page 48).
- `IPDDomNodeExt` interface is exported by every object that exports `IPDDomNode` (see "IPDDomNodeExt Methods" on page 46).
- `IPDDomElement` defines additional methods that apply only to structure elements (see "IPDDomElement Methods" on page 50).
- `IPDDomWord` defines additional methods that apply only to individual words in the document (see "IPDDomWord Methods" on page 52).

Clients of these interfaces must include the files `AcrobatAccess.h`, `AcrobatAccess_i.c` and `IPDDom.h`.

IPDDomNode Data Types

Type	Syntax	Description
`CPDDomNodeType`	`typedef enum { CPDDomNode_Document = 1, CPDDomNode_Page = 2, CPDDomNode_StructElement = 3, CPDDomNode_Text = 4, CPDDomNode_Word = 5, CPDDomNode_Char = 6, CPDDomNode_Graphic = 7, CPDDomNode_Link = 8, CPDDomNode_PushButtonField = 9, CPDDomNode_TextEditField =10, CPDDomNode_StaticTextField =11, CPDDomNode_ListboxField =12, CPDDomNode_ComboboxField =13, CPDDomNode_CheckboxField =14, CPDDomNode_RadioButtonField =15, PDDomNode_SignatureField =16, CPDDomNode_OtherField =17, CPDDomNode_Comment =18, CPDDomNode_TextComment =19, CPDDomNode_Other =20, CPDDomNode_LineSeg =21, CPDDomNode_WordSeg =22 } CPDDomNodeType;`	Defines the type of a node in the PDF DOM hierarchy returned by `GetType`
`PDDom_FontStyle`	`typedef enum { PDDOM_FONTATTR_ITALIC = 0x1, PDDOM_FONTATTR_SMALLCAP = 0x2, PDDOM_FONTATTR_ALLCAP = 0x4, PDDOM_FONTATTR_SCRIPT = 0x8, PDDOM_FONTATTR_BOLD = 0x10, PDDOM_FONTATTR_LIGHT = 0x20 } PDDOM_FontStyle;`	Constants for font styles returned by `GetFontInfo`.
`FontInfoState`	`typedef enum { FontInfo_Unchecked =1, FontInfo_NoInfo =2, FontInfo_MixedInfo =3, FontInfo_Valid =4 } FontInfoState;`	Constants for font status returned by `GetFontInfo`.

Type	Syntax	Description

`DocState`	`enum DocState {`
	`DocState_OK =0,`
	`DocState_Protected =1,`
	`DocState_Empty =2,`
	`DocState_Unavailable =3`
	`};`
		Constants for document status returned by `GetDocInfo` in the `IPDDomDocument` interface.

`NodeRelationship`	`enum NodeRelationship {`
	`NodeRelationship_Descendant =0,`
	`NodeRelationship_Ancestor =1,`
	`NodeRelationship_Before =2,`
	`NodeRelationship_After =3`
	`NodeRelationship_Equal =4,`
	`NodeRelationship_None =5`
	`};`
		Constants returned by `Relationship` in the `IPDDomNodeExt` interface.

IPDDomNode Methods

`IPDDomNode` defines methods that apply to all elements of the document hierarchy.

Method	Syntax	Description
`GetParent`	`LRESULT GetParent (IDispatch **ppDispParent)`	`ppDispParent` returns the `IPDDomNode` for the parent of this element if there ís a parent element in the DOM hierarchy, or `NULL` if this element is the root element of the hierarchy.
`GetType`	`LRESULT GetType (long *nodeType)`	`nodeType` returns the `CPDDomNodeType` of this element.
`GetChild`	`LRESULT GetChild (ASInt32 index, IDispatch **ppDispChild)`	`ppDispChild` returns the `IPDDomNode` for the child of this element at position `index`, or `NULL` if there is no child at position `index`. For a text node, this returns child words; see "Words and Lines in Text" on page 45.
`GetChildCount`	`LRESULT GetChildCount (long *pCountChildren)`	`pCountChildren` returns the number of children of this element
`GetName`	`LRESULT GetName (BSTR *pszName)`	`pszName` returns the name of this element. ● For individual words, this is `NULL` ● For form fields, it is the short description associated with the field. ● For comments, it is a combination of the comment type and subject (if any).
`GetValue`	`LRESULT GetValue (BSTR *pszValue)`	`pszValue` returns the value of this element. ● For individual words, this is the word itself. ● For form fields, it is the current text content of the field. ● For links, it is a description of the associated action. ● For comments, it is the contents. ● For a signature field, it is the name of the signer and the date signed.

Method	Syntax	Description
`IsSame`	`LRESULT IsSame (IPDDomNode *pNode, BOOL *isSame)`	If `pNode` refers to the same node as this element, `isSame` returns `true`.
`GetTextContent`	`LRESULT GetTextContent (BSTR *pszText)`	`pszText` returns the value of all text in the document subtree rooted at this element. Alternate text, actual text, and expansions attributes are included and may override text within the document.
`GetFontInfo`	`LRESULT GetFontInfo (long* fontStatus, BSTR* pszName, float* fontSize, long* fontAttr, float* red, float* green, float* blue)`	These values describe the font characteristics for the text content of this element. ● `fontStatus` returns a value of type `FontInfoState`. —If value is `FontInfo_NoInfo`, the text is not rendered, so it has no font characteristics. Alternate text has no font characteristics, for example. —If value is `FontInfo_Valid`, the rest of the values describe the font characteristics for all of the text in the subtree. That is, each word of the text either has these characteristics or has no font characteristics. —If value is `FontInfo_MixedInfo`, different words of the text have different font characteristics, and the document subtree must be examined more closely to determine which text has which font characteristics. ● `pszName` returns the name of the font. ● `fontSize` returns the point size. ● `fontAttr` returns the set of `PDDom_FontStyle` values. `red`, `green`, `blue` return the RGB components of the color of the text. Each component is a value between 0 and 1.

Method	Syntax	Description
`GetLocation`	`LRESULT GetLocation (long *pxLeft, ong *pyTop, long *pcxWidth, long *pcyHeight)`	Returns the the screen coordinates of the upper left corner, width, and height of the content of the element. Note that this is not exactly the same as the bounding box. If the element spans multiple pages, returns only the location on the first visible page. If none of the elementís contents are visible, returns an empty location.
`GetFromID`	`LRESULT GetFromID (BSTR id, IDispatch **ppDispNode)`	`ppDispNode` returns the `IPDDomNode` for the element in the same document with the matching ID attribute, or `NULL` if there is no such element. The `id` value is not the same as the UID returned by `IAccID` in the MSAA interface; it is an optional attribute of the PDF file itself, as returned by `GetID` in `IPDDomElement`.
`GetIAccessible`	`LRESULT GetIAccessible (IDispatch **ppIAccessible)`	Returns the MSAA `IAccessible` element corresponding to this element. (Acrobat exports an MSAA interface to the document, as well as a DOM interface.) Not all DOM elements have corresponding MSAA elements, because the DOM tree breaks the content down into much smaller pieces. If `ppIAccessible` is `NULL`, search for an ancestor with a non-`NULL` value for `GetIAccessible` to find the corresponding MSAA interface. Use the method `get_PDDomNode` to find the `IPDDomNode` corresponding to a PDF Document `IAccessible` object.
`ScrollTo`	`LRESULT ScrollTo()`	Makes the contents of the node visible. If the contents cover more than one page, only the contents on the first page are made visible. If the entire contents does not fit, the upper left portion is shown.

Method	Syntax	Description
`GetTextInLines`	`LRESULT GetTextInLines (BOOL visibleOnly, IDispatch** ppDispTextLines)`	**`ppDispTextLines`** returns an **`IPDDomNode`** whose children (obtained by calling **`GetChild`**) have the role **`CPDDomNode_LineSeg`**; see "Words and Lines in Text" below. **`visibleOnly`** controls whether the children include only lines that contain at least some visible text. If the role the node is not **`CPDDomNode_Text`**, this method returns **`E_FAIL`**. .

Words and Lines in Text

An **`IPDDomNode`** that represents a text node has the role **`CPDDomNode_Text`**. By default, the children of text nodes are word nodes. To get the word children of a text node, call the **`IPDomNode`** method **`GetChild`**. An **`IPDDomNode`** that represents a word has the role **`CPDDomNode_Word`**.

NOTE: When a word is hyphenated and thus appears on two lines, each segment of the word is returned as a child that has the role **`CPDDom_WordSeg`**.

Text can also be thought of as having lines as children. To get the line children of a text node, call the **`IPDomNode`** method **`GetTextInLines`**. This method returns a new object for the text node. Subsequently, calling **`getChild`** on this object returns lines as children. An **`IPDDomNode`** that represents a line has the role **`CPDDomNode_LineSeg`**. The children of that line node will be the words in that line.

IPDDomNodeExt Methods

The `IPDDomNodeExt` interface is exported by every object that exports `IPDDomNode`. For Acrobat 7.0 and later, the following methods are available from all objects:

Method	Syntax	Description
`Navigate`	`HRESULT Navigate( long navDir, IPDDomNode* next);`	Traverses to another user interface element within a container and retrieves the object. `navDir` indicates which type of navigation is desired, and the node in that direction is returned in `next`. This method is defined in the `IPDDomNodeExt` interface on any node.
`ScrollToEx`	`HRESULT ScrollToEx( BOOL favorLeft, BOOL favorTop);`	Determines where to scroll when the item is too large to fit in the window. If both parameters are `true`, this method is equivalent to `ScrollTo`. This method is defined in the `IPDDomNodeExt` interface on any node.
`SetFocus`	`HRESULT SetFocus();`	Sets the focus to this node, if it can take focus. This method is defined in the `IPDDomNodeExt` interface on any node.
`GetState`	`HRESULT GetState( long* state);`	Returns a set of state flags identical to those returned by `get_accState` for the corresponding `IAccessible` object. This method is defined in the `IPDDomNodeExt` interface on any node.
`GetIndex`	`HRESULT GetIndex( long* pIndex);`	Returns the child index of this node in its parent. The root node returns -1. This method is defined in the `IPDDomNodeExt` interface on any node.
`GetPageNum`	`HRESULT GetPageNum( long* firstPage, long* lastPage);`	Returns the first and last pages on which the node appears. This method is defined in the `IPDDomNodeExt` interface on any node.

Method	Syntax	Description
`DoDefault-Action`	`HRESULT DoDefaultAction();`	Executes the default action for a node. This method is defined in the **`IPDDomNodeExt`** interface on any node.
`Relationship`	`HRESULT Relationship( PDDomNode* node, long* pRel);`	Returns the relationship of the **`node`** parameter to this node. The value is of type **`NodeRelationship`**, defined in IPDDom.h.This method is defined in the **`IPDDomNodeExt`** interface on any node.

IPDDomDocument Methods

The root object for the page or document exports the `IPDDomDocument` interface. For Acrobat 7.0 and later, the following methods are available from the root object:

Method	Syntax	Description
`SetCaret`	`HRESULT SetCaret( IPDDomWord* node, long index);`	Sets the caret to the specified index in the word. If the index is 0, it is placed at the beginning of the word.
`GetCaret`	`HRESULT GetCaret( long* pxLeft, long* pyTop, long* pcxWidth, long* pcyHeight, IPDDomNode** node, long* index);`	Returns the screen location of the caret, the node containing the caret, and the zero-based index of the caret within the node. The node may be a word node or a form field. If there is no active caret, the call returns `S_FALSE`.
`NextFocusNode`	`HRESULT NextFocusNode( BOOL forward, IPDDomNode* node);`	Gets the next or previous focusable `IPDDomNode`. If `forward` is `true`, it gets the next focusable node. Returns `NULL` if there is not another focusable node in the selected direction. Searches only the current DOM tree, which means that in page mode it will only return results within the page tree instead of the entire document.
`GetFocusNode`	`HRESULT GetFocusNode( IPDDomNode* node);`	Returns the `IPDDomNode` with focus. The node is set to `NULL` if the focus is on the document (rather than an annotation) or if the focus is not within the document.
`SelectText`	`HRESULT SelectText( IPDDomWord* startNode, long startIndex, IPDDomWord* endNode, long endIndex);`	Sets the text selection by identifying the start and end locations of the selection.
`GetText-Selection`	`HRESULT GetTextSelection( BSTR* selection);`	Retrieves the value of the selected text.
`GetSelection-Bounds`	`HRESULT GetSelectionBounds( IPDDomWord** start, long* startIndex, IPDDomWord** stop, long* stopIndex);`	**Not implemented.** This procedure always returns `S_FALSE`.

Method	Syntax	Description
`GetDocInfo`	`HRESULT GetDocInfo( BSTR* fileName, long* nPages, long* firstVisiblePage, long* lastVisiblePage, long* status, BSTR* lang);`	Returns the full pathname of the file, how many pages it contains, and the range of pages that are at least partially visible. The `status` indicates whether there are issues with this document or page, such as access controls prohibiting access or an apparently empty page or document. If `lang` is not `NULL`, it is the default language used in the document. **NOTE:** See note below.
`GoToPage`	`HRESULT GoToPage( long pageNum);`	Positions the document so that the requested page is visible. **NOTE:** See note below.

NOTE: The `GetDocInfo` and `GoToPage` methods use different numbering systems. The page numbers returned as `firstVisiblePage` and `lastVisiblePage` by `GetDocInfo` are based on page 1 as the first page of the document. However, the `GoToPage` method treats page 0 as the first page of the document. Therefore, you must adjust accordingly when passing the value of `pageNum` to `GoToPage`.

IPDDomElement Methods

`IPDDomElement` defines additional methods that apply only to structure elements.

Method	Syntax	Description
`GetTagName`	`LRESULT GetTagName (BSTR *pszTagName)`	`pszTagName` returns the structural element tag for this element.
`GetStdName`	`LRESULT GetStdName (BSTR *pszStdName)`	`pszStdName` returns the standard role for this element. The standard roles are: `Document, Part, Art, Sect, Div, BlockQuote, Caption, TOC, TOCI, Index, NonStruct, Private, Table, TR, TH, TD, L, LI, Lbl, LBody, P, H, H1, H2, H3, H4, H5, H6, Span, Quote, Note, Reference, BibEntry, Code, Link, Figure, Formula,Form` For details, see the *PDF Reference, version 1.6*, section 10.7.3.
`GetID`	`LRESULT GetID (BSTR *pszId)`	`pszId` returns the ID string associated with this element, if it has been defined. The `id` value is not the same as the UID returned by `IAccID` in the MSAA interface; it is an optional attribute of the PDF file itself. See Table 10.10 of section 10.6 of the *PDF Reference, version 1.6*.

Method	Syntax	Description
`GetAttribute`	`LRESULT GetAttribute (BSTR pszAttr, BSTR pszOwner, BSTR *pszAttrVal)`	**`pszAttrVal`** returns the value of the specified attribute for specified owner for this element. Owner can be **`NULL`** or an empty string. If the element does not have the requested attribute, the method returns **`S_FALSE`**. The set of owners and attributes is open-ended, but the standard structure attributes for Tagged PDF are defined in section 10.7.4 of the *PDF Reference, version 1.6*. Accessibility attributes are defined in section 10.8. See "Accessibility Attributes" below.

Accessibility Attributes

Some of the attributes that are useful for assistive technology are listed here. For a complete list, see section 10.8 of the *PDF Reference, version 1.6*.

Attribute	Owner	Value
`Lang`		ISO language code for text within this element.
`Alt`		Text containing an equivalent replacement for the content of this element. Automatically incorporated into the value or text content of the element or any of its ancestor elements.
`ActualText`		Text which is an exact replacement for the content of this element, for example., the text of an Illuminated Character. Automatically incorporated into the value or text content of the element or any of its ancestor elements.
`E`		The expanded form of the element's content, when it is an abbreviation or acronym.
`RowSpan`	**`Table`**	Number of rows spanned by the table cell.
`ColSpan`	**`Table`**	Number of columns spanned by the table cell.
`Headers`	**`Table`**	Array of IDs of Table Header (**TH**) cells associated with this table cell (**TD** or **TH**).
`Scope`	**`Table`**	The scope of this table header cell: **`Row`**, **`Column`**, or **`Both`**.

IPDDomWord Methods

`IPDDomWord` defines additional methods that apply only to individual words in the document.

Method	Syntax	Description
`LastWordOfLine`	`LRESULT LastWordOfLine (BOOL *isLast)`	If this is the last word in a line on the page, **`isLast`** returns **`true`**. Use this function to determine where the line breaks occur in text. Note that line breaks are inserted into the text content for elements.

4 Reading PDF Files on Adobe Reader for UNIX

Adobe Reader version 7 for UNIX platforms provides accessibility support for assistive technology such as screen readers. This support is based on the following technologies:

- The Accessibility Tool Kit (ATK). The complete ATK API is documented at http://developer.gnome.org/doc/API/2.0/atk/atk.html.
- The Assistive Technology Service Providers' Interface (AT-SPI) provides accessibility facilities as part of the GNOME project. See http://developer.gnome.org/doc/API/2.0/at-spi/index.html.
- GNOME is a desktop environment that is part of the GNU project for free software. The following web sites provide useful information regarding GNOME accessibility:
  - The GNOME Accessibility Project Website http://developer.gnome.org/projects/gap/.
  - GNOME Accessibility for Developers http://developer.gnome.org/projects/gap/guide/gad/index.html
  - GNOME Accessibility Architecture http://accessibility.kde.org/developer/atk.php and http://www.sun.com/software/star/images/gnome_access_arch_lg.jpg
- The GIMP Toolkit (GTK) is a widget toolkit that is used by GNOME for creating graphical user interfaces. GTK API documentation is located at http://www.gtk.org/api.
- The GObject system is a library and framework that is the basis for all GTK and GNOME classes. GObject documentation is located at http://developer.gnome.org/doc/API/2.0/gobject/index.html

Adobe Reader uses GTK *stock widgets*. It also leverages the GTK Accessibility Implementation Library (GAIL). GAIL contains implementations of ATK interfaces for the standard GTK widgets, making them accessible.

To make user interface elements accessible, Adobe Reader implements the relevant ATK interfaces. The same method is used to expose the contents of PDF documents opened in Adobe Reader.

All ATK-compliant clients can access the user interface of Adobe Reader as well as the contents of PDF documents. Clients must have the AT-SPI development libraries and headers.

Setting up Adobe Reader for Accessibility

GConf is a system that consists of a set of keys for storing application preferences. The key

```
/desktop/gnome/interface/accessibility
```

determines whether accessibility support is enabled on the system. This key must be set to `true` before using the accessibility features of Adobe Reader. This can be done by issuing the following command:

```
gconftool-2 -s /desktop/gnome/interface/accessibility -t bool true
```

In addition, the user must enable document accessibility in the Reader preferences. These preferences can be found under **Edit > Preferences > Reading**. Checking **Enable Document Accessibility** enables tagging of untagged documents. You can also uncheck **Confirm before tagging documents** to allow minimal interruption.

Navigating the Hierarchy of Accessible Objects

The Adobe Reader application is represented as a hierarchy of Accessible objects, which are AtkObjects. In this hierarchy, the application is at the top and other elements are at lower levels. The hierarchy can be navigated to reach any Accessible object.

There are two main categories of Accessible objects that clients can interact with:

- Adobe Reader user interface items (for example, toolbuttons)
- The content of PDF documents (including text, graphics, form fields, and annotations).

You can obtain the Accessible object for the Reader application by iterating over all Accessible children of the Desktop Accessible object (that is all accessible applications). The following code shows how this can be done.

```
Accessible *desktop;
Accessible *accAcroread = NULL;
int numApps, i;

desktop = SPI_getDesktop(0);
numApps = Accessible_getChildCount(desktop); // number of accessible
applications
for(i = 0; i < numApps; i++)
{
     char *name;
     Accessible *child;
     child = Accessible_getChildAtIndex(desktop, i);
     name = Accessible_getName (child);
     if(strcmp(name, "acroread") == 0)
     {
            accAcroread = child;
            break;
     }
Accessible_unref(child);
}
Accessible_unref(desktop);
```

For each accessible application, its name is determined using the **`Accessible_getName`** method until the name "acroread", representing Adobe Reader, is found.

In this example, **`accAcroRead`** is a pointer to the Accessible object for the Adobe Reader application. This object is at the root of the hierarchy of all Accessible objects in Adobe Reader.

The example uses a few of the methods (beginning with "`Accessible_`") that are available for all Accessible objects and can be used to navigate the hierarchy. These methods are the following:

`char *Accessible_getName (Accessible *obj);`
Gets the name of an Accessible object.

`char *Accessible_getDescription (Accessible *obj);`
Gets the description of an Accessible object.

`Accessible *Accessible_getParent (Accessible *obj);`
Gets an Accessible object's parent container.

`long Accessible_getChildCount (Accessible *obj);`
Get the number of children contained by an Accessible object.

`Accessible *Accessible_getChildAtIndex (Accessible *obj, long int childIndex);`
Gets the Accessible child of an Accessible object at a given index.

`AccessibleRelation **Accessible_getRelationSet (Accessible *obj);`
Gets the set of AccessibleRelation objects which describe this Accessible object's relationships with other Accessible objects.

`AccessibleRole Accessible_getRole (Accessible *obj);`
Gets the UI role of an Accessible object. A UTF-8 string describing this role can be obtained via **`Accessible_getRoleName`**.

`char *Accessible_getRoleName (Accessible *obj);`
Gets a UTF-8 string describing the role this object plays in the UI. This method returns useful values for roles that fall outside the enumeration used in **`Accessible_getRole`**.

`AccessibleStateSet *Accessible_getStateSet (Accessible *obj);`
Returns a pointer to an AccessibleStateSet representing the object's current state.

These methods can be used to identify specific items when navigating the hierarchy. The ATK documentation (see page 53) provides more detailed information.

Accessible Object Interfaces

Each Accessible object can implement one or more of several interfaces. For each interface, there is a method (**`Accessible_is`*`InterfaceName`***) that can be called on an object to determine whether the object implements the interface. For example,

```
SPIBoolean Accessible_isAction(Accessible *obj)
```

determines whether an object implements the Action interface.

A list of interfaces implemented by the standard GTK widgets (in GAIL) can be obtained from the GTK/GAIL documentation (see page 53).

The following interfaces are implemented by Accessible objects in Adobe Reader:

- The Action interface (see "Action Interface" on page 58) is implemented for both user interface elements and PDF document contents.
- The Component interface (see "Component Interface" on page 58) is implemented for both user interface elements and PDF document contents.
- The Text interface (see "Text Interface" on page 58) is implemented for text contents of PDF documents.

NOTE: The following interfaces are not currently implemented by Accessible objects in Adobe Reader: Application, EditableText, Hypertext, Image, Selection, StreamableContent, Table, and Value. Further details on these interfaces can be obtained from the ATK or AT-SPI API documentation (see page 53) .

Each interface specifies methods that can be implemented by an Accessible object. To call any of the methods, you must obtain the GObject that corresponds to the interface. For example, Action interface methods apply to **`AccessibleAction`** objects; Text interface methods apply to **`AccessibleText`** objects, and so on.) To get the GObject, call the method **`Accessible_get`*`InterfaceName`*** on the Accessible object. For example, call **`Accessible_getText`** to obtain the **`AccessibleText`** object for an Accessible object.

The following example determines whether an Accessible object (**`accObj`**) implements the Text interface. It then obtains the **`AccessibleText`** object and gets the number of characters in the object.

```
AccessibleText *accText;
long numChars = 0;

if(Accessible_isText(accObj))
{
     accText = Accessible_getText(accObj);
     numChars = AccessibleText_getCharacterCount(accText);
}
```

Action Interface

These methods enable clients to find the possible actions that can be performed on an object and to perform those actions. The Action interface implements the following methods:

```
long AccessibleAction_getNActions (AccessibleAction *obj);
```

Gets the number of actions associated with an object. It returns 0 for objects on which no action can be performed (for example, for a text paragraph or a toolbutton that is disabled). Currently, there is only one action associated with any Accessible object in Adobe Reader. For example, the action "Press" can be performed on a toolbutton in the UI, a push button in a PDF document, or a text annotation to open it. The action "Jump" can be performed on a link annotation to jump to its target.

```
char *AccessibleAction_getName (AccessibleAction *obj, long int i);
```

Gets the name of the *i*th action invokable on this object.

```
char *AccessibleAction_getDescription (AccessibleAction *obj, long int
i);
```

Gets the description of the *i*th action invokable on this object.

```
SPIBoolean AccessibleAction_doAction (AccessibleAction *obj, long int
i);
```

Invokes the action indicated by the index.

Component Interface

The Component interface implements the following methods:

```
void AccessibleComponent_getExtents (AccessibleComponent *obj, long int
long int *y, long int *width, long int *height, AccessibleCoordType
ctype);
```

Gets the bounding box of a component, which can be a UI element such as a toolbutton or a PDF element such as a text form field.

```
short AccessibleComponent_getMDIZOrder (AccessibleComponent *obj);
```

Gets the Z-order of a component in a window layer.

```
SPIBoolean AccessibleComponent_grabFocus (AccessibleComponent *obj);
```

Causes a particular UI element or PDF element to grab the focus.

Text Interface

The Text interface allows clients to obtain the content of any text element in a PDF document, including text-entry form fields as well as textual page contents. To get the entire content of a page as text, a client should navigate the entire hierarchy of the page and accumulate text content through the functions of the Text interface, as well as the names and descriptions of non-text elements on the page.

```
long AccessibleText_getCharacterCount (AccessibleText *obj);
```

Gets the character count of an AccessibleText object.

```
char * AccessibleText_getText (AccessibleText *obj, long int
startOffset, long int endOffset);
```

Gets a range of text from an AccessibleText object.

```
long AccessibleText_getCaretOffset (AccessibleText *obj);
```

Gets the current offset of the text caret in an AccessibleText object.

```
char * AccessibleText_getTextBeforeOffset (AccessibleText *obj, long int
offset, AccessibleTextBoundaryType type, long int *startOffset, long int
*endOffset);
```

Gets delimited text from an AccessibleText object that precedes a given text offset.

```
char * AccessibleText_getTextAtOffset (AccessibleText *obj, long int
offset, AccessibleTextBoundaryType type, long int *startOffset, long int
*endOffset);
```

Gets delimited text from an AccessibleText object that includes a given text offset.

```
char * AccessibleText_getTextAfterOffset (AccessibleText *obj, long int
offset, AccessibleTextBoundaryType type, long int *startOffset, long int
*endOffset);
```

Get delimited text from an AccessibleText object that follows a given text offset.

```
unsigned long AccessibleText_getCharacterAtOffset (AccessibleText *obj,
long int offset);
```

Gets a character at a given offset.

Accessing the User Interface

To interact with Adobe Reader, the client needs to navigate the hierarchy and locate specific objects. These objects can be identified by their roles and their names.

For example, to press the "Print" toolbutton, the client starts from the Accessible object for the Reader application and searches the hierarchy (using the aforementioned methods) for Accessible objects having the role `SPI_ROLE_TOOL_BAR` (obtained through `Accessible_getRole`). It searches their children for an Accessible object having the required name (obtained through `Accessible_getName`) and the role `SPI_ROLE_PUSH_BUTTON` or `SPI_ROLE_TOGGLE_BUTTON`.

The client should then obtain the corresponding AccessibleAction object (through a call to `Accessible_getAction`) and then make a call to `AccessibleAction_doAction` on it, with the action index 0 (for "Press").

The following table shows the SPI roles that correspond to user interface items in Adobe Reader:

Adobe Reader UI item	Role in SPI
Toolbar	`SPI_ROLE_TOOLBAR`
Toolbutton	`SPI_ROLE_PUSH_BUTTON` or `SPI_ROLE_TOGGLE_BUTTON`
Tool tip	`SPI_ROLE_TOOL_TIP`
Pop-up menu	`SPI_ROLE_POPUP_MENU`
Separator	`SPI_ROLE_SEPARATOR`
Navigation tabs list	`SPI_ROLE_PAGE_TAB_LIST`
Navigation tab	`SPI_ROLE_PAGE_TAB`
Progress bar	`SPI_ROLE_PROGRESS_BAR`
Button drop-down	`SPI_ROLE_POPUP_MENU`
Client-drawn area	`SPI_ROLE_DRAWING_AREA`
Hierarchy tree (for example, bookmarks)	`SPI_ROLE_TREE_TABLE`
Graphic or image	`SPI_ROLE_IMAGE`

Accessing PDF Content

This section describes how clients can interact with the contents of a PDF document that is open in Adobe Reader. To access any Accessible object in a PDF document, a client must navigate to it in the same manner as navigating to a UI element.

The accessible object for an open PDF document has a custom role named "Document". (The role name of an accessible object can be obtained by calling `Accessible_getRoleName`.) Under the document object is a hierarchy of the elements that make up a PDF document: items such as pages, paragraphs, form fields, and links. You navigate this hierarchy in the same manner as described in the previous section.

The following table shows the roles that correspond to various elements in a PDF document.

PDF document item	Role in SPI
Pop-up menu	`SPI_ROLE_POPUP_MENU`

PDF document item	Role in SPI
Table	`SPI_ROLE_TABLE` **NOTE:** The content of tables can be obtained by iterating through their children and using the Text interface. Individual cells of a table have the role `SPI_ROLE_UNKNOWN`.
Paragraph or text run	`SPI_ROLE_TEXT`
Text form fields	`SPI_ROLE_DRAWING_AREA`
Push button	`SPI_ROLE_PUSH_BUTTON`
Check box	`SPI_ROLE_CHECK_BOX`
Radio button	`SPI_ROLE_RADIO_BUTTON`
Combo box	`SPI_ROLE_COMBO_BOX`
List box	`SPI_ROLE_LIST`
Drop list	`SPI_ROLE_POPUP_MENU`
Button drop-down	`SPI_ROLE_POPUP_MENU`
Client-drawn area	`SPI_ROLE_DRAWING_AREA`
Hierarchy tree	`SPI_ROLE_TREE_TABLE`
Graphic or image	`SPI_ROLE_IMAGE`
Text annotations	`SPI_ROLE_TEXT`
Other comments (such as stamps)	`SPI_ROLE_PUSH_BUTTON`
Link or hyperlink	Custom role "Link"
Digital signature	Custom role "Signature"

NOTE: In Adobe Reader 7, text components (including textbox form fields) implement the Text interface but not the EditableText interface.