Java Platform, Standard Edition

Java Accessibility Guide

Release 10
E91077-01
March 2018

ORACLE

Java Platform, Standard Edition Java Accessibility Guide, Release 10

E91077-01

Contents

Preface

This document describes Java Access Bridge, Java Accessibility API (JAAPI), and Java Accessibility Utilities, which enable you to create accessible applications.

Audience

This document is intended for developers who want to create Java applications that are accessible to persons with disabilities. Accessible Java applications are compatible with assistive technologies, such as screen readers, screen magnifiers, speech recognition systems, and refreshable braille displays.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Resources

- `javax.accessibility` package
- How to Support Assistive Technologies in The Java Tutorials (Java SE 8 and earlier)
- `com.sun.java.accessibility.util` package

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Java Accessibility Overview

Java SE provides Java Access Bridge, Java Accessibility API (JAAPI), and Java Accessibility Utilities to enable you to create accessible applications.

Topics

- Java Access Bridge
- Java Accessibility API
- Java Accessibility Utilities
- Pluggable Look and Feel

Java Access Bridge

Java Access Bridge enables certain Java applications and applets to be visible to assistive technologies on Microsoft Windows. See Enabling and Testing Java Access Bridge.

Java Accessibility API

The Java Accessibility API (JAAPI), contained in the `javax.accessibility` package, is one of the core parts of the Java Foundation Classes (JFC). The JFCs are a comprehensive set of graphical user interface components and foundation services designed to simplify deployment of Internet, intranet and desktop applications. JAAPI enables you to create Java applications that are accessible to persons with disabilities. Accessible Java applications are compatible with assistive technologies, such as screen readers, screen magnifiers, speech recognition systems, and refreshable braille displays. The JAAPI makes GUI component information available to assistive technologies, giving users alternative presentation and control of Java applications.

Support for JAAPI is built into Swing components; see How to Support Assistive Technologies in The Java Tutorials (Java SE 8 and earlier).

Java Accessibility Utilities

Java Accessibility Utilites, which is contained in the package `com.sun.java.accessibility.util`, is a set of utility classes that help assistive technologies provide access to GUI toolkits that implement the Java Accessibility API. Java Accessibility Utilities monitor events related to UI components. They also help assistive technologies get additional information about a GUI, such as the current position of the mouse, or the window that currently has focus. See Java Accessibility Utilities Overview.

Pluggable Look and Feel

The Java Foundation Classes implement a Pluggable Look and Feel architecture. This architecture allows non-visual manifestations of a user interface to replace or enhance the visual presentation of an application. The expression of the user interface is separated from the underlying structure and data of each individual component. This is accomplished by separating the user interface of the component from its model. The

model of a component is the structure which encapsulates the state and information that is presented to the user by the user interface. For more information on this architecture, see About the JFC and Swing in The Java Tutorials (Java SE 8 and earlier).

2

Java Access Bridge Overview

Java Access Bridge is a technology that enables Java applications and applets that implement the Java Accessibility API to be visible to assistive technologies on Microsoft Windows systems.

Java Access Bridge is a technology that exposes the Java Accessibility API in a Microsoft Windows dynamic-link library (DLL), enabling Java applications and applets that implement the Java Accessibility API to be visible to assistive technologies on Microsoft Windows systems.

In order for existing assistive technologies available on Microsoft Windows systems to provide access to Java applications, they need some way to communicate with Java Accessibility API. Java Access Bridge supports this communication.

An assistive technology application running on Microsoft Windows (for example a screen reader) communicates with Java Access Bridge DLLs, which in turn communicates with the Java Virtual Machine through Java Access Bridge Java libraries. These Java libraries communicate with Java Accessibility API. Java Accessibility API collects information about what is happening in the Java application, which it forwards to the screen reader through Java Access Bridge.

Topics

3

Enabling and Testing Java Access Bridge

By default, Java Access Bridge is not enabled. Enable it either through the command line or the Windows Control Panel. Test it by running a Java application that uses the Accessibility API.

Topics

Java Access Bridge Supported Java SE Platforms

Java Access Bridge is included with JRE 7u6 and later.

Enabling Java Access Bridge Through the Command Line

Enable Java Access Bridge with the `jabswitch` command.

Run the following command (where `%JRE_HOME%` is the directory of your JRE):

```
%JRE_HOME%\bin\jabswitch -enable
```

Enabling Java Access Bridge Through the Control Panel

Enable Java Access Bridge through the Control Panel with the Ease of Access Center.

1. Click **Start**, select **Control Panel**, then **Ease of Access**, then **Ease of Access Center**. Alternatively, press Windows logo key + U to access the Ease of Access Center.
2. Select **Use the computer without a display**.
3. In the section **Other programs installed**, select the check box **Enable Java Access Bridge** (you may have to scroll down).

Disabling Java Access Bridge

Disable Java Access Bridge with the `jabswitch` command.

Run the following command:

```
%JRE_HOME%\bin\jabswitch -disable
```

> **Note:**
>
> You cannot disable Java Access Bridge through the Windows Ease of Access Center.

Testing Java Access Bridge

Test Java Access Bridge by first installing a supported assistive technology then running a Java application that uses the Accessibility API.

1. Ensure that Java Access Bridge is enabled.
2. Install an assistive technology product that supports Java Access Bridge such as one of the following products:
   - JAWS
   - NonVisual Desktop Access (NVDA)
   - Window-Eyes
   - ZoomText
   - SuperNova
3. Run a Java application that uses the `javax.accessibility` package and ensure that your assistive technology product works properly with it.

Java Access Bridge Tools

Use the `accessinspector` and `accesswalker` tools, which are part of the JRE and JDK, to test Java Access Bridge.

The `accessinspector` tool uses the Java Accessibility Utilities API to examine accessible information about the objects in the Java Virtual Machine. The `accesswalker` tool walks through the component trees in a particular Java Virtual Machine and presents the accessibility hierarchy in a tree view. Find these tools in the JRE's or JDK's `bin` directory.

Minimum Version Requirements of Assistive Technologies on 64-Bit Operating Systems

This topic lists the minimum version requirements of some assistive technologies for 64-bit operating systems.

- **JAWS**: Version 13 and later
- **NVDA**: Version 2011.3 and later
- **SuperNova**: Version 13 and later
- **Window-Eyes**: Version 8.2 and later

- **ZoomText**: Version 10.1.5 and later

> **Note:**
>
> 64-bit applications require 64-bit browsers; in particular, if you use Internet Explorer, ensure it's the 64-bit version.

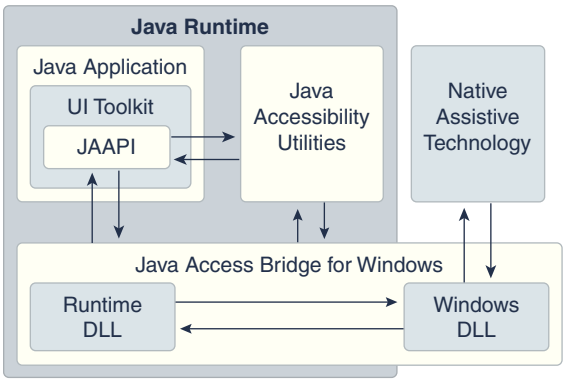
4

Java Access Bridge Architecture

Java Access Bridge consists of a package of classes and DLLs, which enable communication among assistive technologies and Java applications.

The following figure shows how Java Access Bridge and Java Accessibility Utilities components interact with each other:

Figure 4-1 Java Access Bridge Architecture Diagram



Java Access Bridge provides a subset of the Java Accessibility API through the `Windows\System32\windowsaccessbridge-64.dll` Windows DLL. Assistive technologies on Microsoft Windows load and link to this DLL. Java Access Bridge also provides `javaaccessbridge.dll`, which the Java runtime loads. This DLL communicates with the application through the Java Accessibility API and through it, the user interface toolkit and components. The DLL also communicates with the application through Java Accessibility Utilities, a collection of classes that coalesce events and provide application lifecycle functionality to assistive technologies (and to Java Access Bridge, which acts as an assistive technology); see Java Accessibility Utilities Overview. The Java component of Java Access Bridge, manages communication between the DLL loaded into the Java runtime and the other Java code in the Java runtime. The Java component of Java Access Bridge is loaded into the Java SE runtime through what is specified in the `assistive_technologies` property (see Accessibility Properties) and in turn loads the Java-side DLL through Java Native Interfaces (JNI). The communication that Java Access Bridge enables between assistive technologies and Java applications through Java Accessibility Utilities is called interprocess communication.

5

Java Access Bridge API

The Java Access Bridge API enables you to develop assistive technology applications for the Microsoft Windows operating system that work with Java applications. It contains native methods that enable you to view and manipulate information about GUI elements in a Java application, which is forwarded to your assistive technology application through Java Access Bridge.

Topics

Java Access Bridge API Files

The Java Access Bridge API can be found in four files: `AccessBridgeCalls.h` and `AccessBridgeCalls.c` (API calls), `AccessBridgePackages.h` (data structures), and `AccessBridgeCallbacks.h` (callbacks).

Location of Java Access Bridge API Files

Find the following Java Access Bridge API include (header) files in *%JAVA_HOME%* `\include\win32\bridge`:

- `AccessBridgeCallbacks.h`
- `AccessBridgeCalls.h`
- `AccessBridgePackages.h`

Find the file `AccessBridgeCalls.c`, which defines some key interfaces, from the JDK source code repository at `http://hg.openjdk.java.net/jdk9/jdk9/jdk/file/tip/src/jdk.accessibility/windows/native/bridge/AccessBridgeCalls.c`.

Java Access Bridge API Calls

The file `AccessBridgeCalls.h` contains the Java Access Bridge API calls. To use them, compile the file `AccessBridgeCalls.c`. The Java Access Bridge API calls act as the interface between your application and `WindowsAccessBridge.dll`.

Initialization/Shutdown Functions

These two functions start and shut down Java Access Bridge.

- `BOOL initializeAccessBridge();`

Starts Java Access Bridge. You can't use any part of the Java Access Bridge API until you call this function.

- `BOOL shutdownAccessBridge();`

  Shuts down Java Access Bridge. It's important to call this function when your application is finished using Java Access Bridge (before your application exists) so that Java Access Bridge can properly perform memory cleanup.

  Note: Calling the function `shutdownAccessBridge` is not a substitute for releasing any data structures that are maintained by the JVM; do this by calling the function `ReleaseJavaObject`.

Gateway Functions

You typically call these functions before calling any other Java Access Bridge API function:

- `BOOL IsJavaWindow(HWND window);`

  Checks to see if the given window implements the Java Accessibility API.

- `BOOL GetAccessibleContextFromHWND(HWND target, long *vmID, AccessibleContext *ac);`

  Gets the `AccessibleContext` and `vmID` values for the given window. Many Java Access Bridge functions require the `AccessibleContext` and `vmID` values.

Event Handling Functions

These take a function pointer to the function that will handle the event type. When you no longer are interested in receiving those types of events, call the function again, passing in the `NULL` value. Find prototypes for the function pointers you need to pass into these functions in the file `AccessBridgeCallbacks.h`. Java Access Bridge API Callbacks describes these prototypes.

General Functions

- `void ReleaseJavaObject(long vmID, Java_Object object);`

  Release the memory used by the Java object `object`, where `object` is an object returned to you by Java Access Bridge. Java Access Bridge automatically maintains a reference to all Java objects that it returns to you in the JVM so they are not garbage collected. To prevent memory leaks, call `ReleaseJavaObject` on all Java objects returned to you by Java Access Bridge once you are finished with them.

- `BOOL GetVersionInfo(long vmID, AccessBridgeVersionInfo *info);`

  Gets the version information of the instance of Java Access Bridge instance your application is using. You can use this information to determine the available functionality of your version of Java Access Bridge.

> **Note:**
>
> To determine the version of the JVM, you need to pass in a valid `vmID`; otherwise all that is returned is the version of the `WindowsAccessBridge.DLL` file to which your application is connected.

Accessible Context Functions

These functions provide the core of the Java Accessibility API that is exposed by Java Access Bridge.

The functions `GetAccessibleContextAt` and `GetAccessibleContextWithFocus` retrieve an `AccessibleContext` object, which is a magic cookie (a Java `Object` reference) to an `Accessible` object and a JVM cookie. You use these two cookies to reference objects through Java Access Bridge. Most Java Access Bridge API functions require that you pass in these two parameters.

> **Note:**
>
> `AccessibleContext` objects are 64-bit references under 64-bit interprocess communication (which uses the `windowsaccessbridge-64.dll` file). However, prior to JDK 9, `AccessibleContext` objects are 32-bit references under 32-bit interprocess communication (which uses the `windowsaccessbridge.dll` file without `-32` or `-64` in the file name). Consequently, if you are converting your assistive technology applications to run on 64-bit Windows systems, then you need to recompile your assistive technology applications.

The function `GetAccessibleContextInfo` returns detailed information about an `AccessibleContext` object belonging to the JVM. In order to improve performance, the various distinct methods in the Java Accessibility API are collected together into a few routines in the Java Access Bridge API and returned in `struct` values. The file `AccessBridgePackages.h` defines these `struct` values and Java Access Bridge API Callbacks describes them.

The functions `GetAccessibleChildFromContext` and `GetAccessibleParentFromContext` enable you to walk the GUI component hierarchy, retrieving the *n*th child, or the parent, of a particular GUI object.

- `BOOL GetAccessibleContextAt(long vmID, AccessibleContext acParent, jint x, jint y, AccessibleContext *ac)`

  Retrieves an `AccessibleContext` object of the window or object that is under the mouse pointer.

- `BOOL GetAccessibleContextWithFocus(HWND window, long *vmID, AccessibleContext *ac);`

  Retrieves an `AccessibleContext` object of the window or object that has the focus.

- `BOOL GetAccessibleContextInfo(long vmID, AccessibleContext ac, AccessibleContextInfo *info);`

  Retrieves an `AccessibleContextInfo` object of the `AccessibleContext` object `ac`.

- `AccessibleContext GetAccessibleChildFromContext(long vmID, AccessibleContext ac, jint index);`

  Returns an `AccessibleContext` object that represents the *n*th child of the object `ac`, where *n* is specified by the value index.

- `AccessibleContext GetAccessibleParentFromContext(long vmID, AccessibleContext ac);`

  Returns an `AccessibleContext` object that represents the parent of object `ac`.

- `HWND getHWNDFromAccessibleContext(long vmID, AccessibleContext ac);`

  Returns the `HWND` from the `AccessibleContext`of a top-level window.

Accessible Text Functions

These functions get `AccessibleText` information provided by the Java Accessibility API, broken down into seven chunks for efficiency. An `AccessibleContext` has `AccessibleText` information contained within it if you set the flag `accessibleText` in the `AccessibleContextInfo` data structure to `TRUE`. The file `AccessBridgePackages.h` defines the `struct` values used in these functions Java Access Bridge API Callbacks describes them.

- `BOOL GetAccessibleTextInfo(long vmID, AccessibleText at, AccessibleTextInfo *textInfo, jint x, jint y);`
- `BOOL GetAccessibleTextItems(long vmID, AccessibleText at, AccessibleTextItemsInfo *textItems, jint index);`
- `BOOL GetAccessibleTextSelectionInfo(long vmID, AccessibleText at, AccessibleTextSelectionInfo *textSelection);`
- `char *GetAccessibleTextAttributes(long vmID, AccessibleText at, jint index, AccessibleTextAttributesInfo *attributes);`
- `BOOL GetAccessibleTextRect(long vmID, AccessibleText at, AccessibleTextRectInfo *rectInfo, jint index);`
- `BOOL GetAccessibleTextRange(long vmID, AccessibleText at, jint start, jint end, wchar_t *text, short len);`
- `BOOL GetAccessibleTextLineBounds(long vmID, AccessibleText at, jint index, jint *startIndex, jint *endIndex);`

Additional Text Functions

- `BOOL selectTextRange(const long vmID, const AccessibleContext accessibleContext, const int startIndex, const int endIndex);`

  Selects text between two indices. Selection includes the text at the start index and the text at the end index. Returns whether successful.

- `BOOL getTextAttributesInRange(const long vmID, const AccessibleContext accessibleContext, const int startIndex, const int endIndex, AccessibleTextAttributesInfo *attributes, short *len);`

  Get text attributes between two indices. The attribute list includes the text at the start index and the text at the end index. Returns whether successful.

- `BOOL setCaretPosition(const long vmID, const AccessibleContext accessibleContext, const int position);`

Set the caret to a text position. Returns whether successful.

- `BOOL getCaretLocation(long vmID, AccessibleContext ac, AccessibleTextRectInfo *rectInfo, jint index);`

  Gets the text caret location.

- `BOOL setTextContents (const long vmID, const AccessibleContext accessibleContext, const wchar_t *text);`

  Sets editable text contents. The `AccessibleContext` must implement `AccessibleEditableText` and be editable. The maximum text length that can be set is `MAX_STRING_SIZE - 1`. Returns whether successful.

Accessible Table Functions

- `BOOL getAccessibleTableInfo(long vmID, AccessibleContext acParent, AccessibleTableInfo *tableInfo);`

  Returns information about the table, for example, caption, summary, row and column count, and the AccessibleTable.

- `BOOL getAccessibleTableCellInfo(long vmID, AccessibleTable accessibleTable, jint row, jint column, AccessibleTableCellInfo *tableCellInfo);`

  Returns information about the specified table cell. The row and column specifiers are zero-based.

- `BOOL getAccessibleTableRowHeader(long vmID, AccessibleContext acParent, AccessibleTableInfo *tableInfo);`

  Returns the table row headers of the specified table as a table.

- `BOOL getAccessibleTableColumnHeader(long vmID, AccessibleContext acParent, AccessibleTableInfo *tableInfo);`

  Returns the table column headers of the specified table as a table.

- `AccessibleContext getAccessibleTableRowDescription(long vmID, AccessibleContext acParent, jint row);`

  Returns the description of the specified row in the specified table. The row specifier is zero-based.

- `AccessibleContext getAccessibleTableColumnDescription(long vmID, AccessibleContext acParent, jint column);`

  Returns the description of the specified column in the specified table. The column specifier is zero-based.

- `jint getAccessibleTableRowSelectionCount(long vmID, AccessibleTable table);`

  Returns how many rows in the table are selected.

- `BOOL isAccessibleTableRowSelected(long vmID, AccessibleTable table, jint row);`

  Returns true if the specified zero based row is selected.

- `BOOL getAccessibleTableRowSelections(long vmID, AccessibleTable table, jint count, jint *selections);`

  Returns an array of zero based indices of the selected rows.

- `jint getAccessibleTableColumnSelectionCount(long vmID, AccessibleTable table);`

  Returns how many columns in the table are selected.

- `BOOL isAccessibleTableColumnSelected(long vmID, AccessibleTable table, jint column);`

  Returns true if the specified zero based column is selected.

- `BOOL getAccessibleTableColumnSelections(long vmID, AccessibleTable table, jint count, jint *selections);`

  Returns an array of zero based indices of the selected columns.

- `jint getAccessibleTableRow(long vmID, AccessibleTable table, jint index);`

  Returns the row number of the cell at the specified cell index. The values are zero based.

- `jint getAccessibleTableColumn(long vmID, AccessibleTable table, jint index);`

  Returns the column number of the cell at the specified cell index. The values are zero based.

- `jint getAccessibleTableIndex(long vmID, AccessibleTable table, jint row, jint column);`

  Returns the index in the table of the specified row and column offset. The values are zero based.

Accessible Relation Set Function

- `BOOL getAccessibleRelationSet(long vmID, AccessibleContext accessibleContext, AccessibleRelationSetInfo *relationSetInfo);`

  Returns information about an object's related objects.

Accessible Hypertext Functions

- `BOOL getAccessibleHypertext(long vmID, AccessibleContext accessibleContext, AccessibleHypertextInfo *hypertextInfo);`

  Returns hypertext information associated with a component.

- `BOOL activateAccessibleHyperlink(long vmID, AccessibleContext accessibleContext, AccessibleHyperlink accessibleHyperlink);`

  Requests that a hyperlink be activated.

- `jint getAccessibleHyperlinkCount(const long vmID, const AccessibleHypertext hypertext);`

  Returns the number of hyperlinks in a component. Maps to `AccessibleHypertext.getLinkCount`. Returns -1 on error.

- `BOOL getAccessibleHypertextExt(const long vmID, const AccessibleContext accessibleContext, const jint nStartIndex, AccessibleHypertextInfo *hypertextInfo);`

  Iterates through the hyperlinks in a component. Returns hypertext information for a component starting at hyperlink index `nStartIndex`. No more than `MAX_HYPERLINKS` `AccessibleHypertextInfo` objects will be returned for each call to this method. Returns `FALSE` on error.

- `jint getAccessibleHypertextLinkIndex(const long vmID, const AccessibleHypertext hypertext, const jint nIndex);`

  Returns the index into an array of hyperlinks that is associated with a character index in document. Maps to `AccessibleHypertext.getLinkIndex`. Returns -1 on error.

- `BOOL getAccessibleHyperlink(const long vmID, const AccessibleHypertext hypertext, const jint nIndex, AccessibleHypertextInfo *hyperlinkInfo);`

  Returns the *n*th hyperlink in a document. Maps to `AccessibleHypertext.getLink`. Returns FALSE on error.

Accessible Key Binding Function

- `BOOL getAccessibleKeyBindings(long vmID, AccessibleContext accessibleContext, AccessibleKeyBindings *keyBindings);`

  Returns a list of key bindings associated with a component.

Accessible Icon Function

- `BOOL getAccessibleIcons(long vmID, AccessibleContext accessibleContext, AccessibleIcons *icons);`

  Returns a list of icons associate with a component.

Accessible Action Functions

- `BOOL getAccessibleActions(long vmID, AccessibleContext accessibleContext, AccessibleActions *actions);`

  Returns a list of actions that a component can perform.

- `BOOL doAccessibleActions(long vmID, AccessibleContext accessibleContext, AccessibleActionsToDo *actionsToDo, jint *failure);`

  Request that a list of `AccessibleActions` be performed by a component. Returns TRUE if all actions are performed. Returns FALSE when the first requested action fails in which case "failure" contains the index of the action that failed.

Utility Functions

- `BOOL IsSameObject(long vmID, JOBJECT64 obj1, JOBJECT64 obj2);`

  Returns whether two object references refer to the same object.

- `AccessibleContext getParentWithRole (const long vmID, const AccessibleContext accessibleContext, const wchar_t *role);`

  Returns the `AccessibleContext` with the specified role that is the ancestor of a given object. The role is one of the role strings defined in Java Access Bridge API Data Stuctures. If there is no ancestor object that has the specified role, returns `(AccessibleContext)0`.

- `AccessibleContext getParentWithRoleElseRoot (const long vmID, const AccessibleContext accessibleContext, const wchar_t *role);`

  Returns the `AccessibleContext` with the specified role that is the ancestor of a given object. The role is one of the role strings defined in Java Access Bridge API

Data Stuctures. If an object with the specified role does not exist, returns the top level object for the Java window. Returns `(AccessibleContext)0` on error.

- `AccessibleContext getTopLevelObject (const long vmID, const AccessibleContext accessibleContext);`

  Returns the `AccessibleContext` for the top level object in a Java window. This is same `AccessibleContext` that is obtained from `GetAccessibleContextFromHWND` for that window. Returns `(AccessibleContext)0` on error.

- `int getObjectDepth (const long vmID, const AccessibleContext accessibleContext);`

  Returns how deep in the object hierarchy a given object is. The top most object in the object hierarchy has an object depth of 0. Returns -1 on error.

- `AccessibleContext getActiveDescendent (const long vmID, const AccessibleContext accessibleContext);`

  Returns the `AccessibleContext` of the current `ActiveDescendent` of an object. This method assumes the `ActiveDescendent` is the component that is currently selected in a container object. Returns `(AccessibleContext)0` on error or if there is no selection.

- `BOOL requestFocus(const long vmID, const AccessibleContext accessibleContext);`

  Request focus for a component. Returns whether successful.

- `int getVisibleChildrenCount(const long vmID, const AccessibleContext accessibleContext);`

  Returns the number of visible children of a component. Returns -1 on error.

- `BOOL getVisibleChildren(const long vmID, const AccessibleContext accessibleContext, const int startIndex, VisibleChildrenInfo *visibleChildrenInfo);`

  Gets the visible children of an `AccessibleContext`. Returns whether successful.

- `int getEventsWaiting();`

  Gets the number of events waiting to fire.

Accessible Value Functions

These functions get `AccessibleValue` information provided by the Java Accessibility API. An `AccessibleContext` object has `AccessibleValue` information contained within it if the flag `accessibleValue` in the `AccessibleContextInfo` data structure is set to `TRUE`. The values returned are `strings (char *value)` because there is no way to tell in advance if the value is an integer, a floating point value, or some other object that subclasses the Java language construct `java.lang.Number`.

- `BOOL GetCurrentAccessibleValueFromContext(long vmID, AccessibleValue av, wchar_t *value, short len);`
- `BOOL GetMaximumAccessibleValueFromContext(long vmID, AccessibleValue av, wchar_ *value, short len);`
- `BOOL GetMinimumAccessibleValueFromContext(long vmID, AccessibleValue av, wchar_ *value, short len);`

Accessible Selection Functions

These functions get and manipulate `AccessibleSelection` information provided by the Java Accessibility API. An `AccessibleContext` has `AccessibleSelection` information contained within it if the flag `accessibleSelection` in the `AccessibleContextInfo` data structure is set to `TRUE`. The `AccessibleSelection` support is the first place where the user interface can be manipulated, as opposed to being queries, through adding and removing items from a selection. Some of the functions use an index that is in child coordinates, while other use selection coordinates. For example, add to remove from a selection by passing child indices (for example, add the fourth child to the selection). On the other hand, enumerating the selected children is done in selection coordinates (for example, get the `AccessibleContext` of the first object selected).

- `void AddAccessibleSelectionFromContext(long vmID, AccessibleSelection as, int i);`
- `void ClearAccessibleSelectionFromContext(long vmID, AccessibleSelection as);`
- `jobject GetAccessibleSelectionFromContext(long vmID, AccessibleSelection as, int i);`
- `int GetAccessibleSelectionCountFromContext(long vmID, AccessibleSelection as);`
- `BOOL IsAccessibleChildSelectedFromContext(long vmID, AccessibleSelection as, int i);`
- `void RemoveAccessibleSelectionFromContext(long vmID, AccessibleSelection as, int i);`
- `void SelectAllAccessibleSelectionFromContext(long vmID, AccessibleSelection as);`

Java Access Bridge API Data Stuctures

The Java Access Bridge API data structures are contained in the file `AccessBridgePackages.h`.

Important Data Structures

There are data structures in this file that you do not need (and can ignore); they are used as part of the inter-process communication mechanism of the two Java Access Bridge DLLs. The data structures of importance are as follows:

```
#define MAX_STRING_SIZE     1024
#define SHORT_STRING_SIZE    256

typedef struct AccessibleContextInfoTag {
  wchar_ name[MAX_STRING_SIZE];        // the AccessibleName of the object
  wchar_ description[MAX_STRING_SIZE]; // the AccessibleDescription of the object
  wchar_ role[SHORT_STRING_SIZE];      // localized AccesibleRole string
  wchar_ states[SHORT_STRING_SIZE];    // localized AccesibleStateSet string
                                       //   (comma separated)
  jint indexInParent                   // index of object in parent
  jint childrenCount                   // # of children, if any
  jint x;                              // screen x-axis co-ordinate in pixels
  jint y;                              // screen y-axis co-ordinate in pixels
  jint width;                          // pixel width of object
  jint height;                         // pixel height of object
```

```
  BOOL accessibleComponent;           // flags for various additional
  BOOL accessibleAction;              // Java Accessibility interfaces
  BOOL accessibleSelection;           // FALSE if this object doesn't
  BOOL accessibleText;                // implement the additional interface
  BOOL accessibleInterfaces;          // new bitfield containing additional
                                      //   interface flags
} AccessibleContextInfo;

typedef struct AccessibleTextInfoTag {
  jint charCount;       // # of characters in this text object
  jint caretIndex;      // index of caret
  jint indexAtPoint;    // index at the passsed in point
} AccessibleTextInfo;

typedef struct AccessibleTextItemsInfoTag {
  wchar_t letter;
  wchar_t word[SHORT_STRING_SIZE];
  wchar_t sentence[MAX_STRING_SIZE];
} AccessibleTextItemsInfo;

typedef struct AccessibleTextSelectionInfoTag {
  jint selectionStartIndex;
  jint selectionEndIndex;
  wchar_t selectedText[MAX_STRING_SIZE];
} AccessibleTextSelectionInfo;

typedef struct AccessibleTextRectInfoTag  {
  jint x;          // bounding recttangle of char at index, x-axis co-ordinate
  jint y;          // y-axis co-ordinate
  jint width;      // bounding rectangle width
  jint height;     // bounding rectangle height
} AccessibleTextRectInfo;

typedef struct AccessibleTextAttributesInfoTag {
  BOOL bold;
  BOOL italic;
  BOOL underline;
  BOOL strikethrough;
  BOOL superscript;
  BOOL subscript;
  wchar_t backgroundColor[SHORT_STRING_SIZE];
  wchar_t foregroundColor[SHORT_STRING_SIZE];
  wchar_t fontFamily[SHORT_STRING_SIZE];
  jint fontSize;
  jint alignment;
  jint bidiLevel;
  jfloat firstLineIndent;
  jfloat leftIndent;
  jfloat rightIndent;
  jfloat lineSpacing;
  jfloat spaceAbove;
  jfloat spaceBelow;
  wchar_t fullAttributesString[MAX_STRING_SIZE];
} AccessibleTextAttributesInfo;

typedef struct AccessibleTableInfoTag  {
  JOBJECT64 caption;  // AccesibleContext
  JOBJECT64 summary;  // AccessibleContext
  jint rowCount;
  jint columnCount;
  JOBJECT64 accessibleContext;
```

```
  JOBJECT64 accessibleTable;
} AccessibleTableInfo;

typedef struct AccessibleTableCellInfoTag  {
  JOBJECT64  accessibleContext;
  jint       index;
  jint       row;
  jint       column;
  jint       rowExtent;
  jint       columnExtent;
  jboolean   isSelected;
} AccessibleTableCellInfo;

typedef struct AccessibleRelationSetInfoTag {
  jint relationCount;
  AccessibleRelationInfo relations[MAX_RELATIONS];
} AccessibleRelationSetInfo;

typedef struct AccessibleRelationInfoTag {
  wchar_t key[SHORT_STRING_SIZE];
  jint targetCount;
  JOBJECT64 targets[MAX_RELATION_TARGETS];  // AccessibleContexts
} AccessibleRelationInfo;


typedef struct AccessibleHypertextInfoTag {
  jint linkCount;                                 // number of hyperlinks
  AccessibleHyperlinkInfo links[MAX_HYPERLINKS];  // the hyperlinks
  JOBJECT64 accessibleHypertext;                  // AccessibleHypertext object
} AccessibleHypertextInfo;

typedef struct AccessibleHyperlinkInfoTag {
  wchar_t text[SHORT_STRING_SIZE]; // the hyperlink text
  jint startIndex;                 // index in the hypertext document where the link
begins
  jint endIndex;                   // index in the hypertext document where the link
ends
  JOBJECT64 accessibleHyperlink;   // AccessibleHyperlink object
} AccessibleHyperlinkInfo;

typedef struct AccessibleKeyBindingsTag {
  int keyBindingsCount; // number of key bindings
  AccessibleKeyBindingInfo keyBindingInfo[MAX_KEY_BINDINGS];
} AccessibleKeyBindings;

typedef struct AccessibleKeyBindingInfoTag {
  jchar character; // the key character
  jint modifiers;  // the key modifiers
} AccessibleKeyBindingInfo;

typedef struct  AccessibleIconsTag {
  jint iconsCount;                           // number of icons
  AccessibleIconInfo iconInfo[MAX_ICON_INFO]; // the icons
} AccessibleIcons;

typedef struct AccessibleIconInfoTag {
  wchar_t description[SHORT_STRING_SIZE]; // icon description
  jint height;                            // icon height
  jint width;                             // icon width
} AccessibleIconInfo;
```

```
typedef struct AccessibleActionsTag {
  jint actionsCount;                                // number of actions
  AccessibleActionInfo actionInfo[MAX_ACTION_INFO]; // the action information
} AccessibleActions;

typedef struct AccessibleActionInfoTag {
  wchar_t name[SHORT_STRING_SIZE]; // action name
} AccessibleActionInfo;

typedef struct AccessibleActionsToDoTag {
  jint actionsCount;                               // number of actions to do
  AccessibleActionInfo actions[MAX_ACTIONS_TO_DO]; // the accessible actions to do
} AccessibleActionsToDo;

typedef struct VisibleChildenInfoTag {
  int returnedChildrenCount;                        // number of children returned
  AccessibleContext children[MAX_VISIBLE_CHILDREN]; // the visible children
} VisibleChildenInfo;
```

Java Access Bridge API Callbacks

The Java Access Bridge API callbacks are contained in the file `AccessBridgeCallbacks.h`. Your event handling functions must match these prototypes.

You must call the function `ReleaseJavaObject` on every `JOBJECT64` returned through these event handlers once you are finished with them to prevent memory leaks in the JVM.

If you are using legacy APIs, then define `ACCESSBRIDGE_ARCH_LEGACY`.

`JOBJECT64` is defined as `jlong` on 64-bit systems and `jobject` on legacy versions of Java Access Bridge. For definitions, see the section `ACCESSBRIDGE_ARCH_LEGACY` in the `AccessBridgePackages.h` header file.

- `typedef void (*AccessBridge_FocusGainedFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_FocusLostFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_CaretUpdateFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_MouseClickedFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_MouseEnteredFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_MouseExitedFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_MousePressedFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_MouseReleasedFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`
- `typedef void (*AccessBridge_MenuCanceledFP) (long vmID, JOBJECT64 event, JOBJECT64 source);`

- typedef
  void (*AccessBridge_MenuDeselectedFP) (long vmID, JOBJECT64 event,
  JOBJECT64 source);
- typedef void (*AccessBridge_MenuSelectedFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source);
- typedef
  void (*AccessBridge_PopupMenuCanceledFP) (long vmID JOBJECT64 event,
  JOBJECT64 source);
- typedef void (*AccessBridge_PopupMenuWillBecomeInvisibleFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source);
- typedef
  void (*AccessBridge_PopupMenuWillBecomeVisibleFP) (long vmID, JOBJECT64
  event, JOBJECT64 source);
- typedef void (*AccessBridge_PropertyNameChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source, wchar_t *oldName, wchar_t
  *newName);
- typedef void (*AccessBridge_PropertyDescriptionChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source, wchar_t *oldDescription,
  wchar_t *newDescription);
- typedef void (*AccessBridge_PropertyStateChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source, wchar_t *oldState,
  wchar_t *newState);
- typedef void (*AccessBridge_PropertyValueChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source, wchar_t *oldValue,
  wchar_t *newValue);
- typedef void (*AccessBridge_PropertySelectionChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source);
- typedef
  void (*AccessBridge_PropertyTextChangeFP) (long vmID, JOBJECT64 event,
  JOBJECT64 source);
- typedef void (*AccessBridge_PropertyCaretChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source, int oldPosition, int
  newPosition);
- typedef void (*AccessBridge_PropertyVisibleDataChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source);
- typedef
  void (*AccessBridge_PropertyChildChangeFP) (long vmID, JOBJECT64 event,
  JOBJECT64 source, JOBJECT64 oldChild, JOBJECT64 newChild);
- typedef void (*AccessBridge_PropertyActiveDescendentChangeFP)
  (long vmID, JOBJECT64 event, JOBJECT64 source, JOBJECT64 oldActiveDescendent,
  JOBJECT64 newActiveDescendent);

Troubleshooting Java Access Bridge

This topic describes known problems and usage tips for those developing Assistive Technology applications for Java Access Bridge.

Known Problems

Re-Registering Menu Events Generates Duplicate Copies: If you register a menu event, unregister it, and then register it again, then Java Access Bridge generates duplicate copies of the menu event.

`MenuDeselected` Events Generated When Menu is Closed: You are not receiving `MenuCanceled` (or `PopupMenuCanceled`) events. To determine that a menu has been closed, look for `MenuDeselected` events.

Usage Tips

Determining Changes in Menu Item Selection: Use State `PropertyChange` events to determine changes in menu item selection (for example, when the user uses the arrow buttons or keys to go up or down within a menu).

Tracking Values of GUI Elements: Use the `AccessibleValue` support and Value `PropertyChange` events to track the values of GUI elements like sliders and scroll bars.

Determining Selected Items: Use the `AccessibleSelection` support to determine which items are selected in containers that contain items such as lists and tables. This is more efficient than enumerating all of the children and examining their `StateSet` attribute to see if the Selected value is among them.

Java Access Bridge Testing Tools: The Java Access Bridge testing tools `jaccessinspector` and `jaccesswalker` are located in the Java `bin` directory.

6
Accessibility Properties

The `javax.accessibility` package provides the following properties: `assistive_technologies` and `screen_magnifier_present`.

Topics

Loading Assistive Technologies

The `assistive_technologies` property specifies the assistive technologies to load into the JVM. It takes a comma-delimited list of service provider names. See the `javax.accessibility` package, the `javax.accessibility.AccessibilityProvider` abstract class, and the `java.awt.Toolkit.getDefaultToolkit` method.

Indicating the Presence of a Screen Magnifier

When the `screen_magnifier_present` property is set to `true`, it lets the Java platform libraries know that a screen magnifier is present on the system. Application developers can check this property, and if a screen magnifier is present, developers should make sure their applications are compatible with screen magnification. For example, on Microsoft Windows operating systems, the reference implementation of the Java 2D API checks this property and if true, turns off Microsoft DirectDraw to avoid problems with the screen magnifier. (Some screen magnifiers may not be able to magnify DirectDraw graphics.)

Setting Properties

Set a property at run time with the following command:

```
java -Djavax.accessibility.assistive_technologies=ServiceProviderName
```

`ServiceProvicerName` is the name of a service provider that adds an assistive technology feature; see the `javax.accessibility.AccessibilityProvider.getName` method.

You can also specify properties in a file named `.accessibility.properties` in the user's home directory or a file named `accessibility.properties` in the `$JAVA_HOME/conf` directory. In the former case, the properties are used for the current user, and in the latter case, the properties are used for all users of that Java installation. The properties set for the current user take precedence over the properties set for the Java installation.

Set a property in the `accessibility.properties` file by adding line or lines as follows:

```
assistive_technologies=ServiceProviderName
screen_magnifier_present=true
```

> **Note:**
>
> You can specify more than one service provider in the `assistive_technologies` property with a comma-delimited list.

7

Java Accessibility Utilities Overview

To provide access to a Java application, an assistive technology requires more than the Java Accessibility API; it also requires support for locating user interface (UI) objects that implement the Java Accessibility API, loading assistive technology support into the JVM, and tracking events. The Java Accessibility Utilities provide this assistance.

The Java Accessibility Utilities, which is contained in the package `com.sun.java.accessibility.util`, provide the necessary support for assistive technologies to locate and query UI objects inside a Java application running in a JVM. It also provides support for installing event listeners into these objects. These event listeners enable UI objects to learn about specific events occurring in other UI objects using the peer-to-peer approach defined by the delegation event model. This package is made up of the following major pieces:

- Key Information about Java Applications
- Automatic Loading of Assistive Technologies
- Event Support

Key Information about Java Applications

The `com.sun.java.accessibility.util` package contains methods for retrieving key information about Java applications running in a JVM. This support provides a list of the top-level windows of all of the Java applications; an event listener architecture to be informed when top-level windows appear (and disappear); and means for locating the window that has the input focus, locating the mouse position, and inserting events into the system event queue.

Automatic Loading of Assistive Technologies

For an assistive technology to work with a Java application, load it into the same JVM as the Java application to which it is providing access. This is done through the use of the `assistive_technologies` property; see Loading Assistive Technologies. This support is in the class `EventQueueMonitor`.

Event Support

The Java Accessibility Utilities include three classes for monitoring events in the Java Virtual Machine. The first class, `AWTEventMonitor`, provides a way to monitor all AWT events in all AWT components running in the JVM. This class essentially provides system-wide monitoring of AWT events, registering an individual listener for each AWT event type on each AWT component that supports that type of listener. Thus, an assistive technology can register a "Focused listener" with `AWTEventMonitor`, which will in turn register a "Focused listener" with each and every AWT component in each and every Java application in the JVM. Those individual listeners will funnel the events that they hear about to the assistive technology that registered the listener with `AWTEventMonitor` in the first place. Thus, whenever a component gains or loses focus (for example, the user presses the Tab key), the assistive technology will be notified.

The second class, `SwingEventMonitor`, extends `AWTEventMonitor` to provide additional support for monitoring the Swing events supported by the Swing components. Since `SwingEventMonitor` extends `AWTEventMonitor`, there is no need to use both classes if you are using `SwingEventMonitor` in your assistive technology.

The third class, `AccessibilityEventMonitor`, provides support for property change events on `Accessible` objects. When an assisitive technology requests notification of `Accessible` property change events using `AccessibilityEventMonitor`, the `AccessibilityEventMonitor` will automatically register `Accessible` property change listeners on all the components. In addition, it will detect when components are added and removed from the component hierarchy and add and remove the property change listeners accordingly. When an `Accessible` property change occurs in any of the components, the `AccessibilityEventMonitor` will notify the assistive technology.