

Bindows Accessibility, Developer's Guide



Created: May 31, 2006
Author: Joakim Andersson
Update Date: May 31, 2006
Revision No: 1.5

Table of Contents

Introduction	3
General accessibility guidelines	3
Enable accessibility mode.....	4
Screen readers	5
Window-Eyes	5
JAWS.....	5
Narrator.....	5
Magic	5
Custom components.....	7
Built in components.....	9
Inline component.....	10
Known problems	10
Forms mode.....	10
Iframes.....	10
Magnifiers	10
Screen readers	11

Introduction

Bindows is the first framework which provides good accessibility support for rich and thin web applications. The goal has been to minimize the developer effort and make it possible to use existing Bindows applications and make them accessible with only a few settings and we think that we have succeeded.

This document describes how accessibility is handled in Bindows. It's targeted at developers who are using Bindows and wish to make their applications accessible. It also describes how developers can make their own custom components accessible in the same way as the built in components.

General accessibility guidelines

The first step in making an accessible web application based on Bindows is to make sure that everything in the application can be accessed without using the mouse. Most of the built in components work well with the keyboard but to be able to access some of the components you have to set the `tabIndex` property on them. You should make sure that tabbing through your application is logical to the user.

The difference compared to a normal application is that you also have to set tab index to texts, images, progress bars and status bar panels. The reason for this is that screen readers only read information about focused components and to be able to focus on a component, tab index have to be bigger than 0.

If you have images in your application that the user needs to know about, set a tab index to them and add information about the images in the `alt` property of `BiImage`.

Tool bars are not accessible by the default. The reason for this is that tool bars should only be shortcuts to options which are available in the menus. Therefore all commands that are accessible with the mouse from the tool bar should also be accessible from the menu in some way. If you want the tool bar to be keyboard accessible you can add tab index to all of the buttons in the tool bar and it will work.

Internal windows are not accessible by themselves in an application, since they act as focus roots for the components added to them. The built in way to make internal windows accessible is to add them to a `BiDesktopPane`. Then you will be able to enter the first open child window by pressing `CTRL+F6` after tabbing to the desktop pane, and to get back to the desktop pane by pressing `SHIFT+CTRL+F6` in a window. If you want to add windows to another component there are some steps that you will have to take to make them accessible.

- Keep track of the open child windows.
- Add an event listener to the component that listens for `CTRL+F6`, to set focus to the first open window.
- Make sure your component can be reached using the keyboard, i.e. set `tabIndex > 0`.

- Add a description to your component that informs the user of how to enter and exit child windows. For information on how to do this, see custom components section below.

If your controls have descriptive text associated with them you should connect them with the `labelFor` property. The label should not have a tab index since it's read as part of the component. The `labelFor` property can be set like this:

```
<Label text="First name" labelFor="firstNameTxt"/>
<TextField id="firstNameTxt"/>
```

Finally, if something happens in the application that requires the user's immediate attention, create an alert with this information. If you don't do this and add the information to the status bar instead, the user will not hear this information until tabbed to and this may never happen.

Enable accessibility mode

When keyboard navigation works you may also want to make the application accessible for the visually impaired. This is done by adding titles to the different components in the application and these titles are read by screen readers. The titles are created automatically in Bindows but you have to turn accessibility mode on to enable it. You can enable accessibility mode in the ADF like described below or by calling `application.setAccessibilityMode(true)`.

```
<?xml version="1.0"?>
<Application
  accessibilityMode="true"
  accessibilityDescription="Welcome to this application">
  ...
```

When you are in accessibility mode the theme will change a bit. This is due to the fact that some screen readers adds their own interpretation to some colours and this will have strange side effects. You can always check if you are in accessibility mode by calling `application.getAccessibilityMode()`.

When the application starts you can provide the users with a description of the application. This description is shown in an alert and it can be used to inform users of the steps that are needed to get into the correct mode (more information about modes in the next section). The good thing about this alert is that it is definitely read by all screen readers (except Magic).

You can add an accessibility description in the ADF like described below or by calling `application.setAccessibilityDescription(...)`.

```
<?xml version="1.0"?>
<Application
  accessibilityMode="true"
  accessibilityDescription="Welcome to this application">
  ...
```

Screen readers

Bindows have been tested with several different screen readers and for the developers to understand why Bindows behaves the way it does they need to be aware of how the different screen readers work. The more advanced screen readers have a specific mode which is used to read web pages, but since Bindows is mainly used to create real applications, which just happen to run inside a web browser, this mode doesn't suit us very well.

A general guideline about screen readers is to include important information about how they work in the accessibility description.

Window-Eyes

Window-Eyes works very well with Bindows. The default mode is **MSAA mode** and to use Bindows, MSAA mode has to be turned off. This can be done by pressing CTRL+SHIFT+A when the application is started.

JAWS

JAWS is one of the more powerful screen readers and it has a lot of built in commands for navigation. Unfortunately this also means that these commands are reserved for JAWS and Bindows cannot use them. This is why CTRL+TAB will not work for tab panes, CTRL+UP will not work in lists and possibly some other problems may occur as well.

JAWS has to be in **forms mode** to work with Bindows. To achieve this, the user has to enter a text field and press ENTER. In Bindows accessibility mode an invisible text field is created at start up and is put as the first element on the page. The user can press CTRL+INS+HOME to reach this field and then press ENTER to enter forms mode. A big problem with JAWS is that this command doesn't always work.

Another problem that JAWS users have to be aware of is that iframes can turn forms mode off automatically. This happens if the setting *Forms mode auto off* is checked. This can only be solved with configuration files.

Narrator

Narrator is the built in screen reader in Windows. It may not be the best screen reader around but for Bindows it's a perfect match. You don't need to enter any specific mode just start tabbing and enjoy the titles.

Magic

Magic is a combined screen reader and magnifier. The screen reader doesn't have to be in any specific mode to work but it has some problems which cannot be solved in a good way.

The first problem is that the content of alerts isn't read. This is a big problem since the accessibility description will not be read and if you add alerts for important events, these will not be read.

The second problem is that the user will not get any indication that he's currently on a text field, password field, checkbox or radio button. All the other screen readers have built in functionality for these controls and if we add our own description these will hear the description twice.

The final problem is the menus. It works most of the times but on some submenus, not all, it reads all the items instead of just the selected.

Custom components

If you create your own components you probably want them to be accessible in the same way as the built in components. The best way to describe how to do this is by showing an example.

We'll create a custom check box which uses two images to show the current state. To make it as simple as possible the state is changed by pressing any key when the check box has the focus. This component itself will not be described more than by giving the source code. Notice that a change event is thrown when the state of the check box changes.

```
/**
 * A simple custom check box based on two images which is used to
 * illustrate how accessibility can be added to custom components.
 */
function CustomCheckBox() {

    BiComponent.prototype.call(this);
    this._image = new BiImage("images/checkbox_false.gif");
    this.add(this._image);
    this.setTabIndex(1);
    this.addEventListener("keydown", function () {
        this.setChecked(!this.getChecked());
    }, this);
};

_p = CustomCheckBox.prototype = new BiComponent();

/**
 * Property that holds the value of the checkbox. A change event
 * is thrown when the value is changed.
 */
_p._checked = false;
_p.getChecked = function () {
    return this._checked;
};
_p.setChecked = function (bChecked) {
    if (bChecked != this._checked) {
        this._checked = bChecked;
        if (bChecked)
            this._image.setUri("images/checkbox_true.gif");
        else
            this._image.setUri("images/checkbox_false.gif");
        this.dispatchEvent(new BiEvent("change"));
    }
};
```

There are two built in features that can be useful when creating accessible components. Both of these features requires the component to be focusable (that's why tab index is set to 1).

The first feature is used when the component gets focused. In this case a description of the component should be read. This is achieved by implementing the `getDescription` function

which is inherited from `BiComponent`. By implementing this function and returning a description of the component, this description will be read by the screen readers when the component is focused. The implementation can look like this:

```
_p.getDescription = function () {  
    var description = "custom checkbox, ";  
    if (!this.isChecked())  
        description += "not ";  
    description += "checked";  
  
    return description;  
};
```

The second feature can be used when something has to be read after the component is focused but something has happened. In our example we want to read the new state of the check box when the state changes. To achieve this you should implement the `getChangeDescription` function from `BiComponent`. This is done in the same way as for `getDescription`.

```
_p.getChangeDescription = function () {  
    var description = "";  
    if (!this.isChecked())  
        description += "not ";  
    description += "checked";  
  
    return description;  
};
```

The final step is to decide when the change description should be read. In our case we want it to be read when the state of the check box changes. To do this we override the `initAccessibility` function of `BiComponent`. The default implementation of this function handles the focus event and read the description for the component. Therefore you should always call `initAccessibility` of the super class when overriding it.

To initialize the change description we add an event listener and tell it to call `this._on508Change`. This function calls `getChangeDescription` and some other functions to make sure that the text is read by the screen readers.

```
_p.initAccessibility = function () {  
    BiComponent.prototype.initAccessibility.call(this);  
    this.addEventListener("change", this._on508Change);  
};
```

_on508change is a name that will change before release of this. We need a public method with a more descriptive name, for example readChangeDescription or onChangeDescription.

If you want to be able to associate labels with your component, you should add code for this as well. Let's rewrite the getDescription to handle this:

```
_p.getDescription = function () {  
    var description = "";  
  
    var label = this.getForLabel();  
    if (label != null)  
        description += label.getDescription() + " ";  
  
    description += "custom checkbox, ";  
    if (!this.getChecked())  
        description += "not ";  
    description += "checked";  
  
    return description;  
};
```

Built in components

Most of the components in Bindows are accessible out of the box and we have tried to make them work as in any other Windows application. However in some cases you may not agree on the text that we read or you may wish to be more verbose.

In this case you can override the functions described in the previous section (getDescription and getChangeDescription) with your own implementation. If you want to override it for all objects of a specific class, like the status bar panel, you can do it like this:

```
BiStatusBarPanel.prototype.getDescription = function () {  
    return "Status bar = " + this.getText();  
};
```

If you want, you can of course also do it for a single instance of a class like this:

```
progressStatusPanel.getDescription = function () {  
    return "Progress status " + this.getText();  
};
```

Inline component

Bindows can also be used on a normal web page through inline components. Accessibility mode will work in the same way as for a normal Bindows application but you should be careful when you are switching back and forward between pages since this can turn forms mode off and MSAA mode on and even though Bindows is embedded in a normal HTML page you still need to be in the correct mode for accessibility to work.

You should also be extra careful with tab index. All of the elements on the page are handled by Bindows so you should make sure that every component has the correct tab index and that you are able to tab through the entire page.

Known problems

Most things in Bindows work very well in accessibility mode but there are still some problems which developers have to be aware of. Here is a list of the most important problems:

Accessing menus

In normal applications ALT alone opens the menu bar and focuses on the first menu root. As a Bindows application runs inside a browser that browser may have its own menu. To distinguish between the browser menu and the Bindows menu CTRL+ALT is used. However ALT+mnemonic still opens the Bindows menu.

Forms mode

JAWS forms mode is not stable. The problem is that the keyboard command to go to the first form field doesn't always find all the form fields on the page even though can see it. This seems to be a JAWS bug and we haven't been able to go around it yet.

Iframes

Iframes doesn't work with JAWS. The problem is that if an iframe is created after the user have entered forms mode, forms mode will be turned off again. The cause of this problem is a setting in JAWS called *Forms Mode Auto Off* and we haven't found a way to go around this setting without using a configuration file. The problem also applies to BiRichEdit since it extends BiIframe.

Magnifiers

The accessibility work has been focused on screen readers but magnifiers are of course also an import part. Magnifiers work by magnifying the area around the focused element. This works in most cases in Bindows but in some cases is doesn't.

First of all, menus don't work. The problem is that the menu itself cannot retrieve focus and if we move the label which we use read the text in the menu (which is the focused component) to the location of the active menu item it may be hidden by the menu (since menus are always on top) and then it will not be read by screen readers.

Most of the other components work if they are not too big. You may run into problems if you for instance have a large list and move the selection, then the selected item may be outside the visible area since the focus is on the component and not on the selected item.

Screen readers

There is no way to determine which screen reader the user is using and since they all work in slightly different ways it's impossible to make it perfect for all of them but we have tried to make the compromise as good as possible. Most of the screen readers have small problems and these are described in detail in the screen readers section above.