



Federal Aviation Administration Software Applications and Operating Systems

Participant Guide

Table of Contents

	Page
FAA Software Applications and Operating Systems Participant Guide	ii
Part 1: Access Board Software Applications and Operating Systems Technical Standards	1-1
Part 2: Accessibility Features of Software Applications and Operating Systems Java, Unix (Gnome), Microsoft (MSAA)	2-1
Software Applications and Operating Systems Resources	3-1

FAA Software Applications and Operating Systems Participant Guide

Introduction

Overview of Participant Guide

The Participant Guide provides the specific documents and resources information presented during the Software Applications and Operating Systems training. Copies of the Power Point slides utilized during the training are included. Also provided are the specific documents relating to the accessibility features of prevalent software applications and operating systems utilized by software programmers and developers.

Materials are provided for the following software applications and operating systems:

- Java
- Unix (Gnome Accessibility Project)
- Microsoft (Microsoft Active Accessibility (MSAA))

The instructor will direct you through the power Point slides and the resource materials provided in the participants guide during the training.

Elements of the Guide

The Participant Guide includes:

- Software Applications and Operating Systems Power Point Slides
- Participants Notes
- Software Applications and Operating Systems Information and Materials

Software Applications and Operating Systems Participant Guide

Welcome to the FAA Software Applications and Operating Systems Training.

- Each participants should have signed the registration list and received a name tag
- The FAA Software Applications and Operating Systems training will focus on the Section 508 Software Applications and Operating Systems technical standards developed by the Access Board (1194.21).
- This training will present the technical standards and provisions required for the accessibility of FAA software applications and operating systems by individual with disabilities who utilize assistive technology.
- This training will provide technical examples and explanations of how to make inaccessible software applications and operating systems accessible to users of assistive technology.
- The examples will highlight the development of a six-function calculator, named SF Calculator. SF Calculator is created using Visual Basic 6 Professional (VB6Pro), a very popular language and development tool. VB6Pro was selected because it is so widely used, and because, while it is a powerful tool, it is relatively easy to understand and use
- A copy of VB6Pro is required to crate and manipulate the program examples
- This training will provide an overview of the accessibility features of prevalent software applications and operating systems utilized by software programmers and developers. They are: Java, UNIX (Gnome) and Microsoft Active Accessibility (MSAA).
- It is important for software programmers and developers understand how to apply the accessibility features of these widely used software applications and operating systems for ensuring compatibility with assistive technology devices.

The Software Applications and Operating Systems training will present :

- The application of the Section 508, 1194.21, requirements for Software Applications and Operating Systems Technical Provisions.
- A program for a six-function calculator is developed, named SF Calculator. The accessibility requirements of 1194.21 are illustrated during the development of the SF Calculator.
- Provide an Overview of the Accessibility Features of the Java Software Application
- Provide an Overview of the Gnome Accessibility Project and how it relates the UNIX operating system
- Provide an Overview of the Accessibility Features of Microsoft Active Accessibility utilized in Microsoft software applications and operating systems.

The goals of this training is that when you're finished with this training module, you should understand:

- The requirements of the Software Applications and Operating Systems Technical Standards (1194.21)
- The application of the software application Visual Basic 6 Professional (VB6Pro) in creating an accessible program, SF Calculator
- The application of the accessibility requirements of the Software Applications and Operating Systems technical standards during the development of SF Calculator
- Assistive Technology and Software Applications and Operating Systems compatibility and interoperability issues.
- The Accessibility Features of the prevalent software applications and operating systems: Java, UNIX (Gnome) and Microsoft Active Accessibility (MSAA)
- Software Applications and Operating Systems resource and information materials available from other sources.

The information provided in this Software Applications and Operating Systems training has been divided into the following parts:

- Part 1: Access Board Software Applications and Operating Systems Technical Standards (1194.21)
- Part 2: Accessibility Features of Software Applications and Operating Systems
- Java, Unix (Gnome), Microsoft (MSAA)

Part 1: Access Board Software Application and Operating Systems Technical Standards

This section of the training outlines will cover the following:

- The application of the Section 508, 1194.21, requirements for Software Applications and Operating Systems Technical Provisions.
- A program for a six-function calculator is developed, named SF Calculator. The accessibility requirements of 1194.21 are illustrated during the development of the SF Calculator.

The major point of this training is that inaccessible software applications and operating systems interfere with users of assistive technology ability to obtain and use information quickly and easily.

- IT should increase the availability of resources to person with disabilities - barrier-free designs opens doors to this greater audience
- Accessible design bridges the digital divide that locks out people from participating in the workforce on the basis of disability
- Increases individual with disabilities ability to work in a professional and supported work environment
- Accessible IT increases the number of individuals with disabilities ability to receive the FAA message
- Accessible design minimizes risks of non-compliance

**Access Board Technical Standards: Software Applications and Operating Systems
1194.21 Provisions**

Section 508 Rule §1194.21 Software Applications and Operating Systems	
<u>Keyboard Access</u>	<p>§1194.21 When software is designed to run on a system that has a keyboard, product functions shall be executable from a keyboard where the function itself or the result of performing a function can be discerned textually.</p> <p>(a)</p>
<u>Accessibility Features</u>	<p>§1194.21 Applications shall not disrupt or disable activated features of other products that are identified as accessibility features, where those features are developed and documented according to industry standards. Applications also shall not disrupt or disable activated features of any operating system that are identified as accessibility features where the application programming interface for those accessibility features has been documented by the manufacturer of the operating system and is available to the product developer.</p> <p>(b)</p>
<u>Input Focus</u>	<p>§1194.21 A well defined on-screen indication of the current focus shall be provided that moves among interactive interface elements as the input focus changes. The focus shall be programmatically exposed so that assistive technology can track focus and focus changes.</p> <p>(c)</p>
<u>Object Information</u>	<p>§1194.21 Sufficient information about a user interface element including the identity, operation and state of the element shall be available to assistive technology. When an image represents a program element, the information conveyed by the image must also be available in text.</p> <p>(d)</p>
<u>Bitmap Images</u>	<p>§1194.21 When bitmap images are used to identify controls, status indicators, or other programmatic elements, the meaning assigned to those images shall be consistent throughout an application's performance.</p> <p>(e)</p>
<u>Text Information</u>	<p>§1194.21 Textual information shall be provided through operating system functions for displaying text. The minimum information that shall be made available is text content, text input caret location, and text attributes.</p> <p>(f)</p>
<u>User Selected Attributes</u>	<p>§1194.21 Applications shall not override user-selected contrast and color selections and other individual display attributes.</p> <p>(g)</p>
<u>Animation</u>	<p>§1194.21 When animation is displayed, the information shall be displayable in at least one non-animated presentation mode at the option of the user.</p> <p>(h)</p>

Software Applications and Operating Systems

<u>Color Coding</u>	§1194.21 (i) Color-coding shall not be used as the only means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.
<u>Color and Contrast</u>	§1194.21 (j) When a product permits a user to adjust color and contrast settings, a variety of color selections capable of producing a range of contrast levels shall be provided.
<u>Flicker Rate</u>	§1194.21 (k) Software shall not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.
<u>Electronic Forms</u>	§1194.21 (l) When electronic forms are used, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.

Application of 1194.21 Software Applications and Operating Systems

Developing Accessible Software

Creating An Accessible Program, *SFCalculator*

This presentation illustrates the application of the Section 508, 36 CFR 1194.21, requirements for Software Applications and Operating Systems, Technical Provisions (a)-(l). A program for a six-function calculator is developed, named *SFCalculator*. The accessibility requirements of 1194.21 are illustrated during the development of *SFCalculator*.

SFCalculator is created using Visual Basic 6 Professional (VB6Pro), a very popular language and development tool. VB6Pro was selected because it is so widely used, and because, while it is a powerful tool, it is relatively easy to understand and use.

A copy of VB6 is required to create and manipulate the program examples.

Creating the Graphical User Interface (GUI)

The first step is to create the user interface for the calculator. The following process creates the GUI interface:

1. The input/output fields are created.
2. The command buttons are created.
3. A menu is created.
4. The program is compiled and checked.

Creating the Input/Output Fields

1. Enter the VB6 Integrated Development Environment (IDE), and select 'Standard EXE.' Enter the properties Window, and change the following properties of the Form:

Name = sfnCalculator
Caption = SFCalculator
Height = 3885
Left = 105
Top = 105
Width = 6000

2. Next, add fourteen controls to sfnCalculator and set their properties.
3. From the IDE Control ToolBox, select a Label, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the Label:

Name = lblEntry1
Caption = Entry &1
Left = 360
Top = 360

UseMnemonic = True

4. From the Control ToolBox, select a TextBox, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the TextBox:

Name = txtEntry1

Left = 1680

MaxLength = 7

Text = 0

ToolTipText = Please enter your first number.

Top = 360

5. From the Control ToolBox, select a second Label, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the second Label:

Name = lblEntry2

Caption = Entry &2

Left = 3000

Top = 360

UseMnemonic = True

6. From the Control ToolBox, select a second TextBox, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the second TextBox:

Name = txtEntry2

Left = 4320

MaxLength = 7

Text = 0

ToolTipText = Please enter your second number.

Top = 360

7. From the Control ToolBox, select a third Label, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the third Label:

Name = lblResult

Caption = R&esult

Left = 3000

Top = 1560

UseMnemonic = True

8. From the Control ToolBox, select a third TextBox, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the third TextBox:

Name = txtResult

Left = 4320
Locked = True
Text = 0
ToolTipText = This field displays the calculated result.
Top = 1560

The two input fields, result field and their respective field labels have been created for SFCalculator.

Creating the Command Buttons

The next step is to create the eight buttons on sfnCalculator that will represent SFCalculator's eight commands or functions. (Only six of the functions involve calculations. The other two pertain to clearing the fields and exiting the program, respectively.)

The following step will create the eight buttons for the calculator

1. From the Control ToolBox, select a CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the CommandButton:

Name = cmdAdd
Caption = &Add
Left = 240
ToolTipText = Adds Entry 1 and Entry 2 when activated.
Top = 1200

2. From the Control ToolBox, select a second CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the second CommandButton:

Name = cmdSubtract
Caption = &Subtract
Left = 1560
ToolTipText = Subtracts Entry 2 from Entry 1 when activated.
Top = 1200

3. From the Control ToolBox, select a third CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the third CommandButton:

Name = cmdMultiply
Caption = &Multiply
Left = 240
ToolTipText = Multiplies Entry 1 and Entry 2 when activated.
Top = 1800

4. From the Control ToolBox, select a fourth CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the fourth CommandButton:

Name = cmdDivide
Caption = &Divide
Left = 1560
ToolTipText = Divides Entry 1 by Entry 2 when activated.
Top = 1800

5. From the Control ToolBox, select a fifth CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the fifth CommandButton:

Name = cmdRandomize
Caption = &Randomize
Left = 240
ToolTipText = Generates two random numbers and assigns them to Entry 1 and Entry 2, respectively, when activated.
Top = 2400

6. From the Control ToolBox, select a sixth CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the sixth CommandButton:

Name = cmdSquareRoot
Caption = S&quareRoot
Left = 1560
ToolTipText = SquareRoots Entry 1.
Top = 2400

7. From the Control ToolBox, select a seventh CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the seventh CommandButton:

Name = cmdClear
Caption = &Clear
Left = 3000
ToolTipText = Clears Entry 1, Entry 2, and Result.
Top = 2400

8. From the Control ToolBox, select an eighth CommandButton, and place it on sfnCalculator. Enter the properties Window, and change the following properties of the eighth CommandButton:

Name = cmdExit
Caption = E&xit
Left = 4320
ToolTipText = Exits SFCalculator when activated.
Top = 2400

Creating the Menu

Next, use the Menu Editor to add a single Pull Down Menu that includes as its Menu Options all eight program commands or functions. This will later allow the creation of the CommandButton versions.

1. Assign the following properties to the PullDownMenu:

Caption = F&unctions
Name = mnuFunctions

2. To mnuFunctions, add the firstMenuOption, the add function, and assign it the following properties:

Caption = &Add
Name = mnuAdd

3. Please be sure to use 'ALT-R' to designate mnuAdd as a MenuOption of mnuFunctions rather than as another PullDownMenu.
4. Follow similar steps to create the remaining seven MenuOptions of mnuFunctions:
 - a. &Subtract/mnuSubtract,
 - b. &Multiply/mnuMultiply,
 - c. &Divide/mnuDivide,
 - d. &Randomize/mnuRandomize,
 - e. S&quareRoot/mnuSquareRoot,
 - f. &Clear/mnuClear,
 - g. E&xit/mnuExit.
5. Save the program and compile it.
6. Run SFCalculator and check its function.

Provided no errors were made, a fully navigable GUI has been created. The Tab key can be used to move among the three TextBoxes and the eight CommandButtons. The Arrow keys allow movement up or down through the eight MenuOptions of the Functions PullDownMenu.

Code Behind the GUI

Now, enter the following code, recompile, and then run SFCalculator.

Code for SFCalculator

The following is the program code and some comments for the Six-Function Calculator, *SFCalculator*.

```
'Declare global variables.  
Dim varNumber1, varNumber2, varResult As Variant  
Dim strMsg As String  
  
'Execute the Menu-Option version of the Add function if  
and when the user activates it, summing Entry #1 and Entry  
#2.  
Private Sub mnuAdd_Click()  
varResult = varNumber1 + varNumber2  
txtResult = varResult  
txtResult.SetFocus  
End Sub  
  
'Execute the Command-Button version of the Add function  
if and when the user activates it, summing Entry #1 and  
Entry #2.  
Private Sub cmdAdd_Click()  
varResult = varNumber1 + varNumber2  
txtResult = varResult  
End Sub  
  
'Execute the Menu-Option version of the Subtract function  
if and when the user activates it, subtracting Entry #2 from  
Entry #1.  
Private Sub mnuSubtract_Click()  
varResult = varNumber1 - varNumber2  
txtResult = varResult  
txtResult.SetFocus  
End Sub  
  
'Execute the Command-Button version of the Subtract  
function if and when the user activates it, subtracting Entry  
#2 from Entry #1.  
Private Sub cmdSubtract_Click()  
varResult = varNumber1 - varNumber2  
txtResult = varResult  
End Sub  
  
Private Sub mnuMultiply_click()  
varResult = varNumber1 * varNumber2  
txtResult = varResult  
txtResult.SetFocus
```

End Sub

```
Private Sub cmdMultiply_click()  
varResult = varNumber1 * varNumber2  
txtResult = varResult  
End Sub
```

```
'Divide Entry #1 by Entry #2.  
Private Sub mnuDivide_click()  
If varNumber2 = 0 Then  
strMsg = MsgBox("Please do not try to divide by zero!")  
txtEntry2.SetFocus  
Exit Sub 'Error trapping  
End If  
varResult = varNumber1 / varNumber2  
txtResult = varResult  
txtResult.SetFocus  
End Sub
```

```
Private Sub cmdDivide_click()  
If varNumber2 = 0 Then  
strMsg = MsgBox("Please do not try to divide by zero!")  
txtEntry2.SetFocus  
Exit Sub  
End If  
varResult = varNumber1 / varNumber2  
txtResult = varResult  
End Sub
```

'Execute the Menu-Option version of Randomize when and if the user activates it, thereby generating two large random numbers and assigning them to Entry #1 and Entry #2.

```
Private Sub mnuRandomize_click()  
Randomize  
varNumber1 = Int(Rnd(9999999) * 1000000) + 1  
varNumber2 = Int(Rnd(9999999) * 1000000) + 1  
txtEntry1 = varNumber1  
txtEntry2 = varNumber2  
End Sub
```

'Execute the Command-Button version of Randomize when and if the user activates it, thereby generating two large random numbers and assigning them to Entry #1 and Entry #2.

```
Private Sub cmdRandomize_click()  
Randomize  
varNumber1 = Int(Rnd(9999999) * 1000000) + 1
```

```
varNumber2 = Int(Rnd(9999999) * 1000000) + 1
txtEntry1 = varNumber1
txtEntry2 = varNumber2
End Sub
```

'Execute the Menu-Option version of the SquareRoot function when and if the user activates it, producing the square root of Entry #1 only.

```
Private Sub mnuSquareRoot_Click()
varResult = Sqr(varNumber1)
txtResult = varResult
txtResult.SetFocus
End Sub
```

'Execute the Command-Button version of the SquareRoot function when and if the user activates it, producing the square root of Entry #1 only.

```
Private Sub cmdSquareRoot_Click()
varResult = Sqr(varNumber1)
txtResult = varResult
End Sub
```

'Execute the Menu-Option version of the Clear function when and if the user activates it, thereby reinitializing the variables and the three TextBoxes.

```
Private Sub mnuClear_Click()
varNumber1 = 0
varNumber2 = 0
varResult = 0
txtEntry1 = 0
txtEntry2 = 0
txtResult = 0
End Sub
```

'Execute the Command-Button version of the Clear function when and if the user activates it, thereby reinitializing the variables and the three TextBoxes.

```
Private Sub cmdClear_Click()
varNumber1 = 0
varNumber2 = 0
varResult = 0
txtEntry1 = 0
txtEntry2 = 0
txtResult = 0
End Sub
```

'Execute the Menu-Option version of the Exit function when

and if the user activates it, thereby exiting the program.

```
Private Sub mnuExit_Click()  
varNumber1 = 0  
varNumber2 = 0  
Unload Me  
End Sub
```

'Execute the Command-Button version of the Exit function when and if the user activates it, thereby exiting the program.

```
Private Sub cmdExit_Click()  
varNumber1 = 0  
varNumber2 = 0  
Unload Me  
End Sub
```

'Validate that Entry #1 is numeric. If it is not, then warn the user, and return input focus to Entry #1.

```
Private Sub txtEntry1_lostfocus()  
If IsNumeric(txtEntry1.Text) = False Then  
strMsg = MsgBox("Please enter your first number.")  
txtEntry1.SetFocus  
End If  
varNumber1 = Val(txtEntry1.Text)  
End Sub
```

'Validate that Entry #2 is numeric. If it is not, then warn the user, and return input focus to Entry #2.

```
Private Sub txtEntry2_lostfocus()  
If IsNumeric(txtEntry2.Text) = False Then  
strMsg = MsgBox("Please enter your second number.")  
txtEntry2.Refresh  
txtEntry2.SetFocus  
End If  
varNumber2 = Val(txtEntry2)  
End Sub
```

'Upon loading the SF Calculator, initialize variables.

```
Private Sub sfnCalculator_load()  
varNumber1 = 0  
varNumber2 = 0  
varResult = 0  
End Sub
```

Check for entry errors, recompile and run *SF Calculator*.

Exploring the Accessibility of *SFCalculator*

The *SFCalculator* has been developed to demonstrate the application of the Section 508 provisions, specifically the Software Applications and Operating Systems provisions, 36 CFR 1194.21(a)-(l). The following discussion identifies how the various provisions apply and have been implemented in the *SFCalculator* program.

Application of the 1194.21 Provisions

Keyboard Access & Object Information, §1194.21(a) & (d)

The first Technical Provision, 1194.21(a) applies to providing keyboard functionality to programs.

Keyboard Access:

(a) When software is designed to run on a system that has a keyboard, product functions shall be executable from a keyboard where the function itself or the result of performing a function can be discerned textually.

For example, when developing software to run on PCs, at least one alternative keyboard method for any function must be available, if that function or its result can be identified with text, (e.g., a screen reader or speech-recognition system). Note that (a) has Technical Provision (d) as a prerequisite:

Object Information:

(d) Sufficient information about a user interface element including the identity, operation and state of the element shall be available to assistive technology. When an image represents a program element, the information conveyed by the image must also be available in text.

However, using the keyboard to execute commands without knowing what those commands are would be pointless; hence, the importance of (d). Therefore the discussion of provision (d) is included in the context of (a) in this section.

In exploring *SFCalculator*, three alternative keyboard methods of executing the various functions will be demonstrated, as well as two methods of obtaining the results. To be sure, none of the keyboard methods preclude using the mouse to operate *SFCalculator*.

While exploring the remainder of this keyboard example only the keyboard and not the mouse should be used. In addition, if available, using a screen reader will increase the benefit of the example.

Tabbing

One keyboard method used is tabbing. Tab around from Entry 1 until the highlight returns to Entry 1. Along the way, reverse direction by pressing 'Shift-Tab,' and then resume the forward direction to Entry 1. (Note that the Enter key must be pressed to execute the selected

function.)

For purposes of this discussion, there are two points to note. First, there is a definitive keyboard means of accessing and executing the program functions, at least those on the form, and also accessing the results. Second, as when moving from control to control, if a screen reader was used, the screen reader identified each control with a unique text label, and the result of any function executed was available via text. As mentioned above, the first and second conditions are related; keyboard access to *SFCalculator* functions wouldn't make sense without knowing what those functions were. Meeting these two conditions complies with requirements (a) and (d).

So, what in our development of *SFCalculator* enabled these accessibility features? First, the Common Control Components, that VB provided, were used. A benefit of these programming components is that important information about them is available to the OS via the Windows Application Program Interface (Win32API), and, in turn, that information is available to assistive technology (e.g., screen readers or speech-recognition systems). In other words, accessibility features or the potential for them is intrinsic to these components. (If certain properties of controls are not set or defaulted to particular values, then some accessibility features are not effective.) Second, in order to use the keyboard method of tabbing, the TabStop property must be set to True, thereby informing the OS and, in turn, the assistive technology that the given control can be tabbed to. The _Click event informs the OS that the given control can be activated via clicking the mouse or pressing the Enter key. Third, each control that was tabbed to is uniquely identified because it was given a unique text value to its Caption property, in the case of the CommandButtons and of the MenuOptions, had associated Labels to the three TextBoxes and set unique text values to their Caption properties.

To illustrate the power of TabStop and a violation of (a), set TabStop to False for txtEntry1. (There wouldn't be a violation if there were at least one enabled keyboard method of accessing and using Entry 1.) Recompile and run *SFCalculator*. Now notice that Entry 1 or txtEntry1 cannot be tabbed to. (For that matter, it cannot be accessed via its AccessKey, 'ALT-1.' See the second keyboard method below. However the AccessKeys for the CommandButtons do work even when TabStop is disabled.) One can, however, access Entry 1 via clicking the mouse over the edit area.

Now, set TabStop to False for the remaining TextBoxes and CommandButtons. Recompile and run *SFCalculator*. Navigation by tabbing is now disabled. Since keyboard access and use any of the three TextBoxes is not possible, *SFCalculator* violates Technical Provision (a) and thus Section 508. Before continuing, be sure to return TabStop to True for every control.

Temporarily remove the text from Caption for cmdAdd, lblEntry1, lblEntry2, and lblResult. By doing so, Technical Provisions (d) and (a) are violated. Tabbing is possible from TextBox to TextBox to the Add CommandButton and it is even possible to execute the Add function by pressing Enter, the lack of text identifiers for these controls renders keyboard access to and use of these elements difficult and incomplete. Without trial and error, the proper functions of the unidentified TextBoxes and the CommandButton are not known. Before continuing, reenter the text values for cmdAdd, lblEntry1, lblEntry2, and lblResult.

AccessKeys

A second keyboard method uses AccessKeys. Navigate the entire form with AccessKeys, instead of tabbing. Press the following keystrokes with a few-seconds break between each execution (ALT-x may be omitted to avoid exiting *SFCalculator*):

KEYSTROKES	ACCESS WHAT PROGRAM COMMAND OR FUNCTION
ALT-2	Entry 2 input field/TextBox
ALT-e	Result read-only field/TextBox
ALT-a	Add CommandButton
ALT-s	Subtract CommandButton
ALT-m	Multiply CommandButton
ALT-d	Divide CommandButton
ALT-r	Randomize CommandButton
ALT-q	SquareRoot CommandButton
ALT-c	Clear CommandButton
SKIPPING ALT-x	Exit CommandButton
ALT-1	Entry 1 input field/TextBox

Thus a second keyboard alternative to a mouse!

For purposes of this discussion, there are two points to note. First, there is a second definitive and quick keyboard means of accessing and executing any of the program functions and accessing the results. Second, each program function was accessed or executed, for those running a screen reader, the screen reader identified each control with a unique text label, and the result of any function executed was available via text. Meeting these conditions complies with requirements (a) and (d) as well.

So, what in development of *SFCalculator* enabled these accessibility features? Besides using Common Control Components and assigning unique text identifiers to each control (as described above), the AccessKeys were created by including in each Caption ‘&’ immediately prior to the designated AccessKey For instance, the Caption for cmdAdd reads ‘&Add,’ thus designating ‘ALT-A’ as the AccessKey to the Add CommandButton.

Remove the ‘&’ from Caption for all eight CommandButtons and recompile. Run *SFCalculator*. The AccessKeys for the CommandButtons do not work. It is possible, however, tab to each command and press Enter to activate a given function. If tabbing and no other keyboard method were available, disabling the AccessKeys would definitely violate (a) and Section 508. Before continuing, reinsert the ‘&’ for each Caption of the CommandButtons.

PullDownMenus, MenuOptions, and HotKeys

A third keyboard method is provided using the Functions PullDownMenu to select and activate any one of the program commands. Explore the Functions PullDownMenu and its MenuOptions. Use the ALT key to access the MenuBar and the Up and Down Arrow keys to pull down the Functions menu and to move up and down through the various MenuOptions. (Using the ALT-U keys is somewhat quicker for activating the Functions PullDownMenu than using the ALT key and then Enter or Down Arrow. Likewise, from the activated Functions PullDownMenu, using the various HotKeys, such as 'a' for Add or 'c' for Clear, might be quicker than arrowing to the desired option and pressing Enter.) Use the ESC key to escape from or cancel the menu.

Execute the arrow down from the Add option to the Exit option. By executing the down arrow once more, Add option is selected, again. Executing an arrow up selects the Exit option a second time.

Hotkey Navigation

Another navigation method is through the use of Hotkeys. From Function, press 'r' to generate two large random numbers. Return to Function, and press 'q' to produce the square root of the random number in Entry 1.

There are two points to note here. First, there is third method of keyboard use for accessing and executing program functions and results. Second, arrowing from MenuOption to MenuOption, while running a screen reader, verbally identifies each control with a unique text label, and the result of any function executed was available via text. Not surprisingly, meeting these conditions complies with requirements 1194.21(a) and (d).

What in the development of *SFCalculator* enabled these accessibility features? Besides the steps described above, the Menu Editor was used to create mnuFunctions, the Functions PullDownMenu, and its MenuOptions (e.g., mnuAdd). The Caption property of mnuFunctions and mnuAdd et al. was given unique text values, including using '&' in the Caption property to designate a unique mnemonic for quick activation of the given MenuOption.

Were the '&' removed from Caption for each MenuOption, the Functions could still be used, moving up or down and pressing Enter on the selected MenuOption. However, ability to press 'c' to activate the Clear command or 's' to execute the Subtract function, would be disabled.

Choice of Keyboard Methods

So, which keyboard method is preferred? In general, providing all three methods gives applications the most flexibility for a variety of users. Some keyboard users prefer to tab. Some rely on AccessKeys for the fastest response. Others are used to using menus. Some users prefer a combination of methods. Users of speech recognition and users of the mouse could benefit as well, since they could choose, among the three paths mentioned, their preferred way of accessing the user interface via their respective input devices, instead of the keyboard. Many Windows applications exhibit this kind of flexibility (e.g., Microsoft Office Products).

Image Identification

Keyboard access to and text identification of application functions and their results are not the only requirements of Section 508 and not the only accessibility features present in *SF Calculator*. Before turning to an exploration and discussion of some more requirements, notice the second part of Technical Provision (d), namely,

When an image represents a program element, the information conveyed by the image must also be available in text.

To make the second requirement in (d) applicable to *SF Calculator*, modify two properties of `cmdClear`:

Picture = PathName\PictureName
Style = 1-Graphic

`PathName` is a placeholder for the name of the drive and path where `PictureName` is located, and `PictureName` is a placeholder for the name of the imaged file being used. If `Style` is left to its default, then the image will not become visible during runtime. For our image, use a `.gif` named `'Clear.gif'` – presumably, some bitmap/drawing of `'Clear.'` (A `.bmp` image may just as well be used.)

Recompile and run *SF Calculator*. The identity of the Clear `CommandButton` is visually evident and available to users of assistive technology. Actually, `cmdClear`'s identity is visually evident for two reasons (or, it should be): The Clear GIF graphically identifies `cmdClear`, and the value of `Caption` identifies `cmdClear` with text. If a screen reader is used, tabbing to `cmdClear` will read the text, it shouldn't sound any different than it did before loading the image. The `Caption` value of `'Clear'` not only identifies `cmdClear` with text, but it also identifies the image with text that is available to assistive technology. If the text is removed from `Caption`, then the Clear `CommandButton` would still be visually identified by the Clear image from the `Picture` property, but it wouldn't be identified textually nor via many assistive technology. Tabbing to `cmdClear`, again, using a screen reader, the best result would be the identification of `'button.'` So, by assigning appropriate text to the `Caption` property of controls with (or without) images set to `Picture` and `1-Graphic` set to `Style`, a means of complying with both parts of Technical Provision (d) has been provided.

Yet, how is text employed to indicate information conveyed by images representing program functions, but not part of controls with `Caption` properties? Suppose, instead of using `cmdAdd` as a program element for *SF Calculator*, `imgAdd` is used. Use an `Image` control with some graphical plus sign, `'Add.bmp,'` as the picture. Since `Image` controls do not have a `Caption` Property, an `AccessKey` cannot be used to activate it directly, nor could it be labeled with text. While `imgAdd` would be graphically identified and sensitive to mouse actions, therefore it would be, identified via text and sensitive to keyboard and mouse actions by means of the `MenuOption`, `mnuAdd`, as described above.

Accessibility Features, §1194.21(b)

Accessibility Features

(b) Applications shall not disrupt or disable activated features of other products that are identified as accessibility features, where those features are developed and documented according to industry standards. Applications also shall not disrupt or disable activated features of any operating system that are identified as accessibility features where the application programming interface for those accessibility features has been documented by the manufacturer of the operating system and is available to the product developer.

Technical Provision (b) consists of two similar requirements and describes two similar situations. What is the same in both requirements is the moratorium against interfering with activated and documented accessibility features, and the difference is what the accessibility feature belongs to, an OS or another application. The first situation is illustrated when an application interferes with active features of such assistive technology as a screen reader (e.g., disabling the announcement of text that appears on a designated control or portion of the screen). The second situation is illustrated when an application interferes with a Windows Accessibility Option that has been engaged (e.g., disabling some effects of 'Use High Contrast' Mode).

Interference with accessibility features can be caused in a variety of ways and thus comes in various forms: One form is keyboard interference, and another is display interference. If an application uses 'ALT-1' to perform a program function and so does the user's assistive technology, there would be a good chance of a keyboard conflict. The application wins the conflict sometimes, while the assistive technology wins other times. When assistive technology keyboard commands override application keyboard commands, good access and use of that application can be just as problematic as if an accessibility feature of the assistive technology were overridden instead. Sure, some assistive technologies include key-bypass functions, but users of assistive technology shouldn't have to decrease productivity by using an extra keystroke that wouldn't have been necessary had such conflicts not occurred in the first place. As noted above in our discussion of 'Keyboard Methods,' if the users' ability to tab is disabled, via disabling the TabStop property, accessibility features of assistive technology can be disrupted (e.g., not announcing the next control) as well as those of applications (e.g., not being able to use some AccessKeys, assuming one categorizes AccessKeys as such features). If the Accessibility Option of 'Use High Contrast' is checked, and an application includes program code that changes some of the GUI to different colors, then a display conflict likely would be caused. The latter is illustrated in the section on Technical Provision (g), and the section on Technical Provision (k) illustrates how flashing objects at a certain frequency can interfere with features of some assistive technology. As noted in our discussion on 'Focus,' Technical Provision (c), using a graphical caret without simultaneously tracking it with the invisible SystemCaret will disrupt various features of many assistive technologies. Even certain sizes of fonts can interfere with at least one screen reader.

Input Focus, §1194.21(c)

Input Focus

(c) A well-defined on-screen indication of the current focus shall be provided that moves among interactive interface elements as the input focus changes. The focus shall be programmatically exposed so that assistive technology can track focus and focus changes.

In *SF Calculator*, as with most Windows applications, the Focus takes different visual forms. Sometimes it takes the form of the SystemCaret, or the blinking vertical bar that commonly used for editing (e.g., in Entry 1 and Entry 2). Sometimes it takes the form of a dotted rectangle that moves Focus from CommandButton to CommandButton when navigating from one to another (e.g., from cmdAdd to cmdSubtract). Sometimes it takes the form of a moving mouse, while sometimes it takes the form of a highlighted MenuOption (e.g., mnuDivide).

The SystemCaret is behind these various forms of the Focus. With Win32API calls, the SystemCaret can be positioned anywhere on the application window, resized and reshaped, created and destroyed, and turned invisible. Thus, being able to move and track the Focus in the background. The latter allows developers to use their own graphical version of the Focus in the foreground while synchronizing it with the SystemCaret in the background. If that synchronization is absent or implemented unreliably, the assistive technology that relies on SystemCaret information will likely perform equally or more unreliably. In such a case, requirement (c) and Section 508 would be violated.

How was this accessibility feature enabled in *SF Calculator*? The Focus was enabled simply by using the Common Control Components that shipped with VB. A benefit of these programming components is that they intrinsically employ the SystemCaret and its related functions (e.g., getCaretPos and SetCaretPos). By doing so, they provide information vital to the reliable performance of the application and of certain assistive technology (e.g., screen readers, magnification software, or speech-recognition systems).

Bitmap Images, §1194.21(e)

Bitmap Images

(e) When bitmap images are used to identify controls, status indicators, or other programmatic elements, the meaning assigned to those images shall be consistent throughout an application's performance.

Inconsistent use of program elements violates good practices in Programming, Usability, UI Design, and Accessible Software Design. The consistent use of program elements is particularly important for accessibility, which is the reason this provision has been included.

For this illustration:

1. Keep the Picture value of 'Clear.gif,' as described in the previous section.
2. Below cmdClear, place an Image control with Picture set to 'Clear.gif., and copy the code from Sub cmdClear_Click() to Sub imgClear_Click().
3. Recompile and run the program.

The meaning of the images is consistent throughout the running of *SF Calculator*. Suppose, however, the two lines that reset txtEntry1 and txtEntry2 are commented out or removed from Sub imgClear_Click(). When the program is recompiled and run, the meaning of the two images is not exactly the same. Clearly, the former complies with (e), while the latter violates it.

Textual Information, §1194.21(f)

Textual Information

(f) Textual information shall be provided through operating system functions for displaying text. The minimum information that shall be made available is text content, text input caret location, and text attributes.

All three of these criteria are present in *SF Calculator*; although, in our previous explorations, only Text Content was highlighted. Text Content was apparent each `CommandButton` or `TextBox` when it received Focus. It was also conspicuous when entering numbers into Entry 1 or Entry 2.

Text-Input or Caret Position is most obvious when Focus is on a standard `TextBox`, or, generically speaking, an input or edit field. It is especially obvious for those who can see or feel using a Tactile Display; the caret is visible and blinks at a regular rate; and it typically moves right one character space to indicate the current insertion point. The user of a screen reader will not hear the caret blink, but she or he will know, at least tacitly, that an insertion point moves along to the right as he or she types. However, users of screen readers can use designated keystrokes to determine Caret Position. For example, pressing ALT-DEL with one popular screen reader will reveal the Caret Position in x,y coordinates.

Text Attributes, at least some of them (e.g., Bold, Italics, common Fonts and rough Size), are obvious to those who can see them. For example, all of the `AccessKeys` in *SF Calculator* have the Underline attribute. Tab to Entry 1 and select the default text of '0,' Highlight is another attribute of the selected text. (Of course, if the Font is ChineseGothic or the TimesNewRoman Font is 5 Point in size, most Americans who can see are only going to realize that the attributes need to be changed favorably.) As with Caret Position, users of screen readers can use designated keystrokes to determine Text Attributes. For instance, pressing Insert-F with one popular screen reader will reveal something like 'Font = MS Sans Serif 11 Point' if the caret is located on text in one of *SF Calculator*'s `TextBoxes`.

So, how were these accessibility features enabled in *SF Calculator*? They were enabled by using the Common Control Components that shipped with VB. A benefit of these programming components is that important information about them is available to the OS via Win32API, and, in turn, that information is available to assistive technology (e.g., screen readers, magnification software, or speech-recognition systems). In particular, these components intrinsically make information about Text Content, Caret Position, and Text Attributes available via the Win32API and thus to assistive technology. For example, all the standard VB controls employed in *SF Calculator* utilize the `SystemCaret` and its related functions (e.g., `GetCaretPos` and `SetCaretPos`), thereby indicating Focus and assisting the user to seamlessly interact with the application.

However, as mention in the section on Focus, there are ways to interfere with these features and hence ways to violate Technical Provision (f) and Section 508. To recap, one way is to use a purely graphical caret (a mere look-alike, if you will) without synchronizing the `SystemCaret` with the look-alike. (In fact, the bitmap or drawn caret may not look like the "real" caret.) Without that `SystemCaret` to track, assistive technology will not work properly. Another way of interfering would be to employ a non-standard font that didn't use all the necessary text character codes recognized by the OS and thus assistive technology.

Even when employing the SystemCaret and standard fonts, certain values of attributes can interfere with the operation of some assistive technology. For example, at least one screen reader has provided false information when FontSize was set to a value greater than 15 Point for Labels. Set FontSize for lblEntry1 to 16 Point and recompile and run SFCalculator. The screen reader in question correctly read 'Entry 1 edit zero.' Tabbing to Entry 2, however, it incorrectly reads 'Entry 1 edit zero.' Even if the screen is refreshed, it consistently misidentifies Entry 2 as Entry 1. TextBoxes and CommandButtons do not seem to be as prone to this problem. (This problem has been known to occur in some Outlook and Word e-mails and documents, respectively.) A FontSize of 8 to 15 Point seems relatively safe, with 10 to 13 Point being optimum.

User Selected Attributes, §1194.21(g)

User Selected Attributes

(g) Applications shall not override user selected contrast and color selections and other individual display attributes.

Violating Technical Provision (g) is easily illustrated. Add the following code to *SFCalculator*:

```
Private Sub txtEntry1_GotFocus()  
Me.BackColor = vbRed  
txtEntry1.BackColor = vbRed  
txtEntry2.BackColor = vbWhite  
txtResult.BackColor = vbBlue  
End Sub
```

Recompile and run *SFCalculator*, to see the difference. (If need be, a screen reader's designated keystrokes can be used to announce the changes in foreground and background colors.) Even without running the modified *SFCalculator*, the appended code shows that the background colors of the Form and the three TextBoxes are red, red, white, and blue, respectively. How patriotic! Putting aside the fact that *txtEntry2* has white text on a white background, an accessibility barrier has been created for individuals with certain types of color/contrast-sensitivity. Surely not, two methods are available, the HighContrast Setting via Accessibility Options can be modified or the Windows Appearance Scheme can be changed, both in the Windows ControlPanel! Both approaches will now be demonstrated.

Activate the Accessibility Options in the ControlPanel, and check the first option on the Display Tab, which reads 'Use High Contrast.' Activate the *OkButton*, and return to run our modified *SFCalculator*. Guess what? While most of Windows changed its appearance, the red, red, white, and blue background colors remain in this *SFCalculator*. (Note that the latter not only illustrates how to violate Technical Provision (g), but it shows how to violate the second part of Technical Provision (b).) By unchecking the 'Use High Contrast' option and select 'High Contrast' from the Appearance Tab of Display in the ControlPanel, the same results are found; the background colors remain. If a person couldn't use applications without their having High Contrast, like white on black, then they couldn't use this version of *SFCalculator*, as well as any other applications that violated (g) in this fashion.

Animation, §1194.21(h)

Animation

(h) When animation is displayed, the information shall be displayable in at least one non-animated presentation mode at the option of the user.

In other words, information expressed in an animation, at the choice of the user, needs to be made available in a non-animated, accessible format. How provision (h) is implement (h) is left to the programmer's creativity. One implementation of (h) would be to have an application provide an option to skip animation, while providing an accessible version of any information conveyed by that skipped animation. Another implementation of (h) would be to give the user the option to display the animation while conveying the information in an accessible, non-animated format.

A simple program, 'Section 508 Animation,' will now be created that will illustrate these two implementations of (h). The code is listed below, but first create the form and other controls, that is the GUI.

Creating the GUI

1. Enter the VB6 Integrated Development Environment (IDE), and select 'Standard EXE.'
2. Enter the properties Window, and change the following properties of the Form:

Name = frmSection508Animation
Caption = Section 508 Animation
Height = 3600
Left = 0
ScaleMode = 3-Pixel
Top = 0
Width = 4800

3. Next, add eight controls to frmSection508Animation and set their properties.
4. From the Control ToolBox, select three CommandButtons, four Labels, and one TextBox, and place them on frmSection508Animation.
5. For each control, enter their properties Window, and change the following properties:

Command1
Name = cmdPressMeFirst
Appearance = 0-Flat
Caption = PressMeFirst!
Left = 136
Top = 152

Command2

Name = cmdPressMeSecond
Appearance = 0-Flat
Caption = PressMeSecond!!
Left = 48
Top = 304

Command3
Name = cmdPressMeFinale
Caption = PressMeFinale!!
Left = 856
Top = 608

Label1
Name = lblLine1
AutoSize = True
Caption =
Left = 48
Top = 40

Label2
Name = lblLine2
AutoSize = True
Caption =
Left = 128
Top = 184

Label3
Name = lblLine3
AutoSize = True
Caption =
Left = 32
Top = 328

Label4
Name = lblStatus
Caption = Status:
Height = 49
Left = 32
Top = 600

Text1
Name = txtStatus
Height = 49
Left = 120
Locked = True
MultiLine = true
Text =

**Top = 600
Width = 721**

Code Behind the GUI

Now, enter the code that is listed below 'Code for Section 508 Animation,' correct any errors, compile, and then run it.

Code for Section 508 Animation

'The following is the code for the program, "Section 508 Animation.exe."

**'Declare global variables
Dim Shift, intPause, intCounter, intLabelCount, intStep,
intCount1, intCount2 As Integer
Dim blnMoveForward, blnMoveBackward As Boolean
Dim strMsg As String**

**'Turns the CommandButtons invisible
Private Sub HideCommandButtons()
cmdPressMeFirst.Visible = False
cmdPressMeSecond.Visible = False
cmdPressMeFinale.Visible = False
End Sub**

**'Creates the yellow-on-blue StatusBar
Private Sub StatusBar()
lblStatus.BackColor = vbBlue
txtStatus.BackColor = vbBlue
lblStatus.ForeColor = vbYellow
txtStatus.ForeColor = vbYellow
lblStatus.FontSize = 10
lblStatus.FontBold = True
End Sub**

**'Displays this text after PressMeFinale vanishes
Private Sub DisplayStatus4()
txtStatus.Text = "After activating PressMeFinale, 'Section
508' scrolled right off the screen; 'Guide To' scrolled left off
the screen; 'Accessible Software' scrolled right; and
PressMeFinale vanished. To exit, click the close symbol at
the top right, or press 'ALT-F4.'"
End Sub**

**'Displays this status when PressMeSecond vanishes to the
right**

```
Private Sub DisplayStatus3()  
txtStatus.Text = "You pushed PressMeSecond to the far  
right. Slightly below, left of where PressMeSecond stopped,  
appeared one-inch, bold white letters, 'Accessible Software.'  
PressMeSecond vanished, and PressMeFinale appeared at  
the bottom right corner."  
cmdPressMeFinale.Visible = True  
End Sub
```

'Displays this status when PressMeFirst returns from its trip and vanishes

```
Private Sub DisplayStatus2()  
txtStatus.Text = "PressMeFirst quickly moved across the  
screen, left to right to left again, leaving a temporary trail,  
then vanished. Slightly above where PressMeFirst  
traversed, 'Section 508' appeared in one-inch, bold white  
letters. Slightly below, 'Guide To' appeared in one-inch,  
bold white letters. Slightly below and left of 'Guide To,'  
appeared a grey button with black text, 'PressMeSecond!'  
Please press SPACEBAR on PressMeSecond until it  
vanishes."  
End Sub
```

'Displays this status immediately following the user selecting the animation path

```
Private Sub DisplayStatus1()  
txtStatus.Text = "On a black background, above and left of  
center, appears a grey button with black text,  
'PressMeFirst!' Across the bottom of the screen, there's a  
blue StatusBar with yellow text, what you're currently  
reading. Please click or press ENTER or SPACE on  
PressMeFirst to begin the animation."  
End Sub
```

'Displays this status text if skipping animation

```
Private Sub DisplayStatus0()  
txtStatus.Text = "On a black background, starting from the  
top, in one-inch bold white letters, appears 'Section 508' (on  
line 1), 'Guide To' (on line 2), and 'Accessible Software' (on  
line 3). Across the bottom of the screen, is a blue StatusBar  
with yellow text, what you're currently reading. To exit,  
click the close symbol at the top right, or press 'ALT-F4.'"  
End Sub
```

'Displays the third line of text, "Accessible Software"

```
Private Sub DisplayLine3()  
lblLine3.BackColor = vbBlack
```

```
lblLine3.ForeColor = vbWhite
lblLine3.FontBold = True
lblLine3.FontName = "Times new Roman"
lblLine3.FontSize = 72
lblLine3.Caption = "Accessible Software"
End Sub
```

```
Private Sub DisplayLine2()
lblLine2.BackColor = vbBlack
lblLine2.ForeColor = vbWhite
lblLine2.FontBold = True
lblLine2.FontName = "Times new Roman"
lblLine2.FontSize = 72
lblLine2.Caption = "Guide To"
End Sub
```

```
Private Sub DisplayLine1()
lblLine1.BackColor = vbBlack
lblLine1.ForeColor = vbWhite
lblLine1.FontBold = True
lblLine1.FontName = "Times new Roman"
lblLine1.FontSize = 72
lblLine1.Caption = "Section 508"
End Sub
```

```
'Pause subroutine, used to slow animation
Private Sub Pause()
For intPause = 1 To 75000
Next intPause
End Sub
```

```
'Initiates movement of text
Private Sub cmdPressMeFinale_Click()
Call Disappear
End Sub
```

```
'Upon each press of SpaceBar, moves PressMeSecond
slightly right
Private Sub cmdPressMeSecond_KeyDown(KeyCode As
Integer, Shift As Integer)
If KeyCode = vbKeySpace And Shift = 0 Then
If blnMoveForward = True Then
intCount1 = intCount1 + intStep
cmdPressMeSecond.Left = intCount1
If intCount1 >= 700 Then
If intLabelCount = 2 Then
Call HideCommandButtons
```

```
Call DisplayLine3
Call DisplayStatus3
intLabelCount = 3
End If
blnMoveForward = False
blnMoveBackward = True
intCount2 = intCount1
End If
End If
End If
End Sub
```

'Upon one press of SpaceBar or Enter, sends PressMeFirst zooming to the right and back, leaving a temporary trail

```
Private Sub cmdPressMeFirst_Click()
SendKeys ("{ENTER}")
Pause
If blnMoveForward = True Then
intCount1 = intCount1 + intStep
cmdPressMeFirst.Left = intCount1
If intCount1 >= 500 Then
If intLabelCount = 0 Then
Call DisplayLine1
intLabelCount = 1
End If
blnMoveForward = False
blnMoveBackward = True
intCount2 = intCount1
End If
End If
If blnMoveBackward = True Then
intCount2 = intCount2 - intStep
cmdPressMeFirst.Left = intCount2
If intCount2 <= 40 Then
If intLabelCount = 1 Then
Call DisplayLine2
intLabelCount = 2
End If
blnMoveForward = True
blnMoveBackward = False
intCount1 = intCount2
cmdPressMeFirst.Visible = False
cmdPressMeSecond.Visible = True
cmdPressMeSecond.SetFocus
End If
End If
Call DisplayStatus2
```

End Sub

'Executes animated vanishing act of "Section 508 Guide To Accessible 'Software'"

Private Sub Disappear()

Call DisplayStatus4

For intCounter = lblLine1.Left To 1100

Pause

lblLine1.Move (intCounter)

Next intCounter

For intCounter = lblLine2.Left To -500 Step -1

Pause

lblLine2.Move (intCounter)

Next intCounter

For intCounter = lblLine3.Left To 1050

Pause

lblLine3.Move (intCounter)

Next intCounter

cmdPressMeFinale.Visible = False

End Sub

'Prepares form for interactive animation, and presents user with a choice...

Private Sub YesAnimation()

cmdPressMeSecond.Visible = False

cmdPressMeFinale.Visible = False

strMsg = MsgBox("Do you wish to display a StatusBar that will convey the animation's information, but in static text?", vbYesNo)

If strMsg = vbYes Then

Call StatusBar

Call DisplayStatus1

End If

If strMsg = vbNo Then

txtStatus.Visible = False

lblStatus.Visible = False

End If

End Sub

'Conveys the informational content of the animation path, without using animation

Private Sub NoAnimation()

Call HideCommandButtons

Call DisplayLine1

Call DisplayLine2

Call DisplayLine3

Call StatusBar

```
Call DisplayStatus0
End Sub
```

```
'Presents the user with a choice...
Private Sub AnimationOrNot()
strMsg = MsgBox("Would you like to skip the animation
(Yes/No)?", vbYesNo)
If strMsg = vbYes Then Call NoAnimation
If strMsg = vbNo Then Call YesAnimation
End Sub
```

```
'Initializes some variables and prepares the form
Private Sub Form_Load()
intLabelCount = 0
intStep = 3
intCount1 = cmdPressMeFirst.Left
intCount2 = intCount1
blnMoveForward = True
blnMoveBackward = False
frm508Animation.WindowState = vbMaximized
frm508Animation.BackColor = vbBlack
Call AnimationOrNot
End Sub
```

Exploring and Running Section 508 Animation

When the program run, the user is prompted to choose between skipping the animation and not skipping the animation. Do not skip it the first time. When the user chooses the animation path, they are given the option of having the animation display without or with a StatusBar that will describe the animation and its information in a static, textual version.

For those of us who can see well enough, the scene should be clear with a glance. Those using a screen reader, should hear something like 'PressMeFirst Button.' If they tab, they should hear text that informs them as to what's present on the screen and what to do. When activating PressMeFirst and PressMeSecond then, by tabbing to the StatusBar, updated information about the animation is heard. The same is true about the StatusBar when the SpaceBar is used to push PressMeSecond far to the right, so that it vanishes, and shortly after being activated PressMeFinale.

This first path illustrates one way of implementing Technical Provision (h) (the second implementation method mentioned above). The option was given to run the animation without or with a non-animated presentation of the animation's information. When the latter is chosen, such a presentation mode was available, along with the interactive animation. If the former path is chosen, the interactive animation and no StatusBar is presented, which would not be fully accessible to those without usable sight.

Rerun Section 508 Animation. This time, choose the shortest path to the animation's information, at least the essential information. When choosing to skip the animation, the three lines of large static text and the StatusBar describing the scene are presented. Arguably, the

essential information conveyed by the animation, when run, was the three lines of large, bold white text. The interactive components and the disappearing lines of text were essential elements and events for the animation, because they drove the animation, but not its basic information.

This shorter path illustrates another way of implementing Technical Provision (h) (the first implementation method mentioned above). The option to skip the animation is available. When animation is skipped, the basic animation information was conveyed to us textually, that is, in an accessible, non-animated presentation mode.

So, how are these two ways of implementing requirement (h) provided? No especially challenging development was required. First, the GUI is created using standard controls with built-in accessibility, that is, information about them available to assistive technology via the Win32API. Only minor problems were experienced in developing the GUI. Hearing the Caption of lblStatus if Height of it or txtStatus was set higher than 49 was impaired when using one popular screen reader, and the Height and Width of txtStatus had to be adjusted to accommodate different content. Second, from the code, it is discerned that relatively straightforward programming was employed. The two vbYesNo MsgBoxes and four If-Thens provide and manage the user options and user's decisions, respectively. The StatusBar and DisplayStatus procedures provide and manage the accessible StatusBar and its content. Basically, these two general parts provide the application features required by (h) and Section 508. The rest of the code generates the interactive animation or is dressing.

Color Coding, §1194.21(i)

Color Coding

(i) Color coding shall not be used as the only means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.

In other words, this Technical Provision requires that, when using color as an indicator, use it in conjunction with a textual indicator.

To illustrate compliance with (i) and then a violation of (i), modify SFCalculator twice. First, in the blocks of code for mnuDivide and cmdDivide, insert the following line of code immediately following the If-Then line:

txtEntry2.BackColor = vbRed

Immediately following the End-If line, insert in both blocks the following line of code:

txtEntry2.BackColor = vbWhite

Recompile and run SFCalculator. When trying to divide by zero, not only do a warning message delivered, but also a secondary warning indicator is received in the form of the BackColor red of Entry 2. Once the division is changed to divide by a non-zero number, the Divided-by-Zero red warning vanishes from Entry 2.

Second, in the same two blocks of code, temporarily comment out the line of code that causes the MessageBox warning. Then recompile and run SFCalculator. Now, when trying to divide by zero, all that occurs is Entry 2 turns red and receives focus. If the user were color blind, he or she wouldn't benefit from any color indicator. Even if the user weren't color blind, some users wouldn't necessarily be able to infer the significance of red without a simple textual warning.

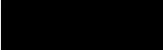
Color and Contrast, §1194.21(j)

Color and Contrast

(j) Color-coding shall not be used as the only means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.

For some people, the use of color is a matter of preference. For others it is a matter of necessity. Some people with vision impairments require high contrast color schemes while others need to have softer, unsaturated colors and less contrast so as not to suffer a visual “white out.” People who suffer eyestrain after even short sessions on the computer find that different color and contrast settings ease the discomfort. The solution to this diversity of requirements is to provide a range of foreground and background color choices. This provision does not require software to provide color and contrast settings. However, products that do provide color and contrast an adjustment, this provision requires a variety of color combinations producing a range of contrast levels.

For most applications support of the operating system color choices for text and background colors will meet this requirement. If the application is not able to inherit user selected system-wide foreground and background color choices, then provide viewing choices that set both background and foreground text colors. At a minimum, use the following 16 color pallet or an equivalent is recommended.

#	Color Name (Color names are per HTML 4.0)	Color	RGB Value (Hexadecimal)
1	Black		#000000
2	Blue		#0000FF
3	Lime		#00FF00
4	Red		#FF0000
5	Aqua		#00FFFF
6	Fuchsia		#FF00FF
7	Yellow		#FFFF00
8	White		#FFFFFF
9	Navy		#000080
10	Green		#008000
11	Maroon		#800000
12	Teal		#008080
13	Purple		#800080
14	Olive		#808000
15	Grey		#808080
16	Silver		#C0C0C0

Flicker Rate, §1194.21(k)

Flicker Rate

(k) Software shall not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.

Technical Provision (k) is intended to prevent applications from inducing seizures by users prone to having them. By complying with (k), interference with normal operations of assistive technology might be prevented as well. Rather than illustrating what can be added to our programs to make our applications more accessible, the following illustrates what should not be included. This application would be in violation of (k) and Section 508.

Blinker is a simple illustration of the kinds of flashing objects that should not include in applications. When the blink rate is between two and fifty-five times per second seizures may be induced in people who are sensitive to photo-induced seizures. An element with a lower or higher blink rate would be in compliance. Before entering the code below 'Code for Blinker,' create the GUI.

Creating the GUI

Start a new Standard.Exe, and set the Name property of Form1 to frmBlinker. On frmBlinker, place Label1 left of center, Text1 in the center, and Command1, Command2, and Command3 left to right below Label1 and Text1. In order assign them, the following Name and Caption values, respectively: lblBlinkingTextBox/Blinking TextBox, txtBlinker, cmdIncreaseBlinkRate/&Increase Blink Rate, cmdDecreaseBlinkRate/&Decrease Blink Rate, and cmdExit/E&xit. Check for errors, compile, and test the GUI.

Code Behind the GUI

Enter the code listed below, check for errors, recompile, and run Blinker.

Code for Blinker

'What follows is the code for the Blinker program. Blinker prompts the user to flash a TextBox between black and white and from zero to one-hundred times per second.

'Declare global variables.

**Dim intBlinkRatePerSecond, intCount As Integer
Dim strWarning As String**

'Flashes txtBlinker black and white every Timer1.interval

Private Sub Timer1_Timer()

If intCount = 1 Then

txtBlinker.BackColor = vbWhite

txtBlinker.ForeColor = vbBlack

End If

If intCount = 0 Then

txtBlinker.BackColor = vbBlack

txtBlinker.ForeColor = vbWhite

End If

```
intCount = intCount Xor 1
End Sub
```

'Increases the blink rate to no more than one hundred when activated

```
Private Sub cmdIncreaseBlinkRate_Click()
If intBlinkRatePerSecond < 100 Then
intBlinkRatePerSecond = intBlinkRatePerSecond + 1
If intBlinkRatePerSecond > 0 Then Timer1.Interval = 500 /
intBlinkRatePerSecond
If intBlinkRatePerSecond = 0 Then Timer1.Interval = 0
txtBlinker.Text = intBlinkRatePerSecond
End Sub
```

'Decreases the blink rate to no less than zero when activated

```
Private Sub cmdDecreaseBlinkRate_Click()
If intBlinkRatePerSecond > 0 Then intBlinkRatePerSecond
= intBlinkRatePerSecond - 1
If intBlinkRatePerSecond > 0 Then Timer1.Interval = 500 /
intBlinkRatePerSecond
If intBlinkRatePerSecond = 0 Then Timer1.Interval = 0
txtBlinker.Text = intBlinkRatePerSecond
End Sub
```

'Exits Blinker when activated

```
Private Sub cmdExit_Click()
Unload Me
End Sub
```

'Initializes variables and instructs/warns user about Blinker

```
Private Sub Form_Load()
intCount = 0
intBlinkRatePerSecond = 0
Timer1.Interval = intBlinkRatePerSecond
txtBlinker.Text = intBlinkRatePerSecond
strWarning = MsgBox("You can use the Increase (ALT-I)
and Decrease (ALT-D)Buttons to flash the TextBox from
zero to one-hundred times per second. Pressing ALT-X will
exit Blinker. If a blink rate between two and fifty-five hertz
will cause a viewer a ceasure, DO NOT increase the blink
rate above two or below fifty-five! Also, a blink rate higher
than nine may interfere with announcing of the
CommandButtons by some screen readers.")
End Sub
```

Exploring Blinker

When Blinker is run, the user is given some instructions and warnings, and then an easy-to-use interface. The user can tab and use Enter, click, or use AccessKeys to access and activate the three CommandButtons. The focus can be placed on the TextBox using the mouse or by tabbing to it. The blink rate can be increased and decreased. Now, try the entire gambit of values while using whatever assistive technology is available (being sure to tab around occasionally between changes). Certain blink rates interfere with some assistive technology.

Blinker primarily an illustration of non-compliance with Section 508,

Electronic Forms, §1194.21(l)

Electronic Forms

(l) When electronic forms are used, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.

Technical Provision (l) requires that forms be compatible with assistive technology and be fully accessible and usable by users of such technology. If keyboard alternatives are provided for navigating through a form, and all elements of the form, including fields to be completed, have sufficiently descriptive text labels located near them, the form is more likely to meet this requirement. Note that, where applicable, no part of the form must violate any of the other requirements, (a)-(k); again, all Technical Provisions must be met for a product to be compliant with Section 508, unless 36 CFR 1194.5 ('Equivalent Facilitation') applies. For example, if (l) is met, but a permanent 30-Hz blinking object displays or it has unchangeable colors somewhere on the form, then the form would be non-compliant in either case.

What is an 'electronic form'? Presumably, it is a software-based version of a paper form; one or more sheets of paper with information/instructions/questions and requiring completion by providing certain information and, once completed, submission. A problem with this working definition is that some forms do not have paper versions, or, if they do, benefits of the electronic versions are lost switching to the paper version. Many electronic forms perform input validation, are very interactive, and, in principle, can be completed and submitted by people with or without disabilities; paper forms do not have these advantages. Take the simple Windows Search Dialogue or any Search form on the Web, as an example. The Windows Search Dialogue provides information/instructions and requires the user to provide input and to submit it in order for it to perform a search; and the same is for web Search forms. If printed out, the paper counterparts are not serachable. Certainly, there are forms that are completed, printed, and mailed, but electronic forms are not limited to having paper counterparts. Accordingly, forms are ubiquitous in the computer world, on the desktop and via the Web. A simple OpenFile or SaveFile Dialogue counts as a form, and so does an online long version of the 1099.

Forms are so varied and so many that providing a guide for creating but a couple is beyond the scope of this document. There are many form-specific software packages as well, and these require their own guides. Some of the guides even address creation of forms with accessibility in mind. The following illustrations create two similar, simple forms, one with VB and the other with Word XP, both of which comply with Technical Provision (l). Both forms have the same name, 'Contact Information Form.'

Creating the VB Version

Create the VB version of 'Contact Information Form.' As usual, begin with creating the GUI. In fact, creating the GUI is the majority of labor behind creating the 'Contact Information Form'. The GUI pretty much is the form. The code provides some instructions and enables the user to print or exit the form.

Creating the GUI

1. Enter VB, and choose a Standard.Exe.
2. Set the following properties of Form1:

Name = frmContactInformation
Caption = Contact Information Form
Height = 8340
Left = -75
Top = -270
Width = 8145

3. From the Control ToolBox, place twelve Labels, twelve TextBoxes, and two CommandButtons onto frmContactInformation.
4. Deposit the controls in the following order, and assign them the following property values:

Label1
Name = lblFirstName
AutoSize = True
Caption = &First Name
Height = 195
Left = 720
Top = 360

Text1
Name = txtFirstName
Height = 405
Left = 720
Text =
Top = 720
Width = 1935

Label2
Name = lblMiddleName
AutoSize = True
Caption = &Middle Name
Height = 195
Left = 2880
Top = 360

Text2
Name = txtMiddleName
Height = 405
Left = 2880
Text =
Top = 720

Width = 1935

Label3

Name = lblLastName

AutoSize = True

Caption = &Last Name

Height = 195

Left = 5040

Top = 360

Text3

Name = txtLastName

Height = 405

Left = 5040

Text =

Top = 720

Width = 1935

Label4

Name = lblAddressLine1

AutoSize = True

Caption = Address Line &1

Height = 195

Left = 720

Top = 1560

Text4

Name = txtAddressLine1

Height = 405

Left = 1800

Text =

Top = 1560

Width = 4935

Label5

Name = lblAddressLine2

AutoSize = True

Caption = Address Line &2

Height = 195

Left = 720

Top = 2160

Text5

Name = txtAddressLine2

Height = 405

Left = 1800

Text =

Top = 2160
Width = 4935

Label6
Name = lblCity
AutoSize = True
Caption = Village/Town/&City
Height = 195
Left = 720
Top = 2880

Text6
Name = txtCity
Height = 405
Left = 720
Text =
Top = 3240
Width = 1935

Label7
Name = lblState
AutoSize = True
Caption = &State
Height = 195
Left = 2880
Top = 2880

Text7
Name = txtState
Height = 405
Left = 2880
Text =
Top = 3240
Width = 1935

Label8
Name = lblCountry
AutoSize = True
Caption = C&ountry
Height = 195
Left = 5040
Top = 2880

Text8
Name = txtCountry
Height = 405
Left = 5040

Text =
Top = 3240
Width = 1935

Label9
Name = lblZipCode
AutoSize = True
Caption = &Zip Code
Height = 195
Left = 720
Top = 3960

Text9
Name = txtZipCode
Height = 405
Left = 720
MaxLength = 9
Text =
Top = 4320
Width = 1095

Label10
Name = lblHomePhone
AutoSize = True
Caption = &Home Phone #
Height = 195
Left = 2880
Top = 3960

Text10
Name = txtHomePhone
Height = 405
Left = 2880
MaxLength = 10
Text =
Top = 4320
Width = 1095

Label11
Name = lblWorkPhone
AutoSize = True
Caption = &Work Phone #
Height = 195
Left = 5040
Top = 3960

Text11

Name = txtWorkPhone
Height = 405
Left = 5040
MaxLength = 10
Text =
Top = 4320
Width = 1095

Label12
Name = lblGlobalID
AutoSize = True
Caption = &Global Identification #
Height = 195
Left = 720
Top = 5040

Text12
Name = txtGlobalID
Height = 405
Left = 2400
Text =
Top = 5040
Width = 2895

Command1
Name = cmdPrint
Caption = &Print
Left = 3840
Top = 6360

Command2
Name = cmdExit
Caption = E&xit
Left = 5400
Top = 6360

5. Correct any errors, compile, and run the form.
6. Here's the test. Try it with some assistive technology. A screen reader will read the name of each field as it is tab or shift-tab to, that is, as it receives Focus. Other assistive technology should be compatible with the form as well.

The Print and Exit functions have not been implemented yet. Enter the code to make them work and to provide the user some instructions.

Code for Contact Information Form

'What follows is the code for the program, "Contact Information Form.exe"

**'Exits and unloads the program when activated
Private Sub cmdExit_Click()
Unload Me
End Sub**

**'Prints the form when activated
Private Sub cmdPrint_Click()
frmContactInformation.PrintForm
End Sub**

**'Displays instructions upon startup
Private Sub Form_Load()
MsgBox "Please complete the following Contact Information Form. Each field has a corresponding AccessKey (e.g., ALT-f for Firstname, ALT-1 for Address line 1, ALT-h for Home Phone #, etc.). Otherwise, use the mouse or TAB to navigate from field to field. There are two CommandButton AccessKeys, ALT-p to Print the form and ALT-x to Exit."
End Sub**

Exploring the Form

Correct any errors, recompile, and run Contact Information Form. The form should work better than before, for the Print and Exit CommandButtons have code to drive them. Again, test it with and without assistive technology.

As discussed in previous sections, there are three general reasons why this form is accessible. First, standard controls are employed. Second, every control has a text identifying it; the TextBoxes have corresponding Labels that have appropriate Caption values, and the CommandButtons have appropriate Caption values. Third, the Labels are either vertically or horizontally aligned with the top left corners of their corresponding TextBoxes.

Creating the Word XP Form

Creating a simple 508-compliant form in Word XP can be done in a few steps. (Refer to 'Contact Information Form.doc' for the sample Word form and more details about the form.)

1. Open a blank, new Word document.
2. Activate the Forms ToolBar on the View Pull-Down Menu.
3. On the first line, center justify and enter the title, 'Contact Information Form.'
4. On the fourth line, enter the first field label, 'First name.'
5. On the next line, right under 'First name,' insert the first EditBox or TextFormField; both terms refer to the same edit-field control that is found on the Forms ToolBar.
6. Press Enter twice.
7. Repeat the third through fifth steps for the remaining field labels (i.e., 'Middle name,' 'Last name,' 'Address Line #1' ... 'Global Identification'). Each field label should appear on the line above the edit field and two lines below the previous edit field.
8. On the Tools Pull-Down Menu, activate the Protect Document option; activating this option allows one to tab from field to field and input, delete, or edit field contents.

Part 2: Accessibility Features of Software Applications and Operating Systems

Java, UNIX (Gnome), Microsoft (MSAA)

This section of the training outlines the following information:

- Overview of the Accessibility Features of the Java Software Application

Sun Microsystems Accessibility Program: Developer Information

Resources For Application Developers:

- Designing for Accessibility - This will assist application developers who are currently not using the Motif toolkit.
- Accessibility Quick Reference Guide – All about accessibility, what it is, why it is important, tips, and additional resources
- Java[tm] Accessibility Quick Tips – Quick tips on how to make Java[tm] applications accessible
- The Java Tutorial – Accessibility Section - Online tutorial on how to use various swing features
- Developing Accessible JFC Applications – A shorter companion document to IBM’s “Accessible Java Programming Guidelines” that focuses on strategies for passing Java[tm] Accessibility Helper tool tests.
- IBM Guidelines for writing Applications Using 100% Pure Java[tm] - Written for application developers
- IBM Software Accessibility Checklist
- IBM Java Accessibility Checklist

JAVA Accessibility Features (www.java.sun.com)

Java Accessibility is currently broken into two separate packages:

The Java Accessibility API

A cross platform, toolkit independent API designed to give assistive technologies direct access to the information in user interface objects and is part of the Java Foundation Classes. Defines a contract between individual user-interface components that make up a Java application and an assistive technology that is providing access to that Java application. If a Java application fully supports the Java Accessibility API, then it should be compatible with, and friendly toward, assistive technologies such as screen readers, screen magnifiers, etc. With a Java application that fully supports the Java Accessibility API, no off screen model would be necessary because the API provides all of the information normally contained in an off screen model.

The Java Accessibility API package consists of 8 Java programming language interfaces, and 6 Java programming language classes:

The Java Accessibility Utilities

In order to provide access to a Java application, an assistive technology requires more than the Java Accessibility API: it also requires support in locating the objects that implement the API as well as support for being located into the Java virtual machine, tracking events, etc. The Java Accessibility utility classes provide this assistance.

The Java Accessibility Utilities are delivered by Sun as a separately downloadable package for use by assistive technology vendors in their products which provide access to Java applications running in a Java Virtual Machine. This package provides the necessary support for assistive technologies to locate and query user interface objects inside a Java application running in a Java Virtual Machine. It also provides support for installing "event listeners" into these objects. These event listeners allow objects to learn about specific events occurring in other objects using the peer-to-peer approach defined by the delegation event model introduced in JDK1.1.

This package is still in active development, and is not as fully defined as the Java Accessibility API. This package is made up of the following major pieces:

Key information about the Java Application(s)

This package contains methods for retrieving key information about the Java application(s) running in the Java Virtual Machine. 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. In order to provide this support immediately for the JDK1.1 environments, Sun took advantage of an implementation detail in the Sun reference implementation of the JDK. In the Sun implementation, the system-wide EventQueue can be replaced with an alternate one. The Java

Accessibility Utilities provide an alternate system EventQueue in the class EventQueueMonitor that implements the functionality described previously.

Automatic Loading of Assistive Technologies

In order for an assistive technology to work with a Java application, it needs to be loaded into the same Java Virtual Machine as the Java application it is providing access to. This is done by extending the class libraries to look for a special configuration line in the awt.properties file specifying a list of assistive technology classes to load. This support is in the class EventQueueMonitor, which is a replacement for the system event queue. As stated above, the EventQueueMonitor implementation is dependent upon specific details of the Sun reference implementation of the JDK1.1 Java Virtual Machine, and not on the formal specification. Because of this, automatic loading of assistive technologies may not work in all JDK1.1 environments. The automatic loading of assistive technologies is part of the JDK1.2 specification, however, so this support will be in all Java Virtual Machines that support JDK1.2.

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 Java Virtual Machine. 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 Java Virtual Machine. Those individual listeners will funnel the events 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 (e.g. the user hits 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 assistive 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.

AWT Translators

With the release of the [Java Foundation Classes](#) (JFC), many developers who were using the AWT to build the user interfaces of their Java applications will switch to the new Swing classes in the JFC. Many will also update their existing AWT programs to Swing. Still, a

significant number of Java applications will remain using some AWT components for displaying their user interfaces. The Java Accessibility Utilities contain a set of classes which implement the Java Accessibility API on behalf of AWT components -- in effect translating for them! These translators work in concert with the support for finding Accessible components in the first place, which is part of the EventQueueMonitor method `getAccessibleAt`. If the object at that location isn't an actual instance of an Accessible, the `getAccessibleAt` method looks for a Translator that will implement the Accessible interface on behalf of that component.

Like much of the rest of the Java Accessibility support, the translator architecture is completely extensible. Any programmer can create a translator. As long as the user's environment is configured properly, the Java Accessibility utility classes will automatically find the new translator and engage it. This means that both mainstream developers and assistive technology vendors can create and distribute new Accessible Translators, making formerly inaccessible user interface components accessible in the process.

Sample Source Code

In addition to the utility classes and translator architecture, the Java Accessibility Utilities includes several example assistive technology programs. The example programs include programs that monitor AWT and Swing events, a program that fully exercises the Java Accessibility API for the component underneath the mouse, and a program that traverses the component hierarchy, displaying the entire hierarchy in tree view.

The Java Accessibility Bridge to Native Code (www.java.sun.com)

Java applications run on a wide variety of host operating systems, many of which already have assistive technologies to provide access to program written in the Java programming language, they need a bridge between themselves in their native environment(s) and the Java Accessibility support that is available from within the Java virtual machine (or Java VM). This bridge, by virtue of the fact that it has one end in the Java VM and the other on the native platform, will be slightly different for each platform it bridges to. Sun is currently developing both the Java programming language side of this bridge, and the Win32 side. In cooperation with assistive technology vendors and the various platform vendors.

In order for existing assistive technologies available on host systems (e.g. Microsoft Windows, Macintosh, OS/2) to provide access to Java applications, they need some way to communicate with the Java Accessibility support in those Java applications. The Java Accessibility Bridge supports that communication. This bridge is a class which contains "native methods." Part of the code for the class is actually supplied by a DLL on the host system - Solaris, OS/2, Microsoft Windows, Macintosh, etc. The assistive technology running on the host (e.g., a Macintosh screen reader) communicates with the Macintosh native DLL portion of the bridge class, which in turn communicates with the Java Virtual Machine, and from there to the Java Accessibility utility support and the Java Accessibility API on the individual user interface objects of the Java application it is providing access to.

For example, in order for a screen reader for Microsoft Windows to provide access to Java applications running on that system, that screen reader would make calls to the Java

Accessibility Bridge for Microsoft Windows. If/when a the user launched a Java application, the bridge would inform the screen reader of this fact. Then the screen reader would query the bridge about the Java application and the bridge would in turn forward those queries on to the Java Accessibility Utilities that were loaded into the Java Virtual Machine, and in many cases on to the individual user interface object that implemented the Java Accessibility API. When those answers came back to the bridge, the bridge would forward them on to the screen reader for Microsoft Windows, which would then use the answers to tell the user what was going on in the Java application

The Pluggable Look and Feel of the Java Foundation Classes (www.java.sun.com)

The Java Foundation Classes (or LFC) are a collection of technologies that represent a new foundation upon which to build Java applications. The Java Accessibility API is one of the technologies of the JFC. Another is the “Swing” set of user-interface components, which is built using Pluggable Look and Feel architecture. This architecture separates the implementation of the user-interface components from their presentation. On a component-by-component basis, the presentation is programmatically determined and can be chosen by the user. Instead of a visual presentation, a user could instead choose an audio presentation, or a tactile (e.g. Braille) presentation, or a combination of the two. With this support, a user would not need a separate assistive technology product interpreting the visual presentation of the program on the screen. Instead, the user would have direct access to that program because it would interact with the user in his/her modality.

How the Swing classes provide a Pluggable Look and Feel

For each component in Swing, there are actually (at least) five Java programming language objects that are needed to make that Swing component pluggable. These are: the component itself (e.g., a button); the Java programming language interface that defines the user interface (e.g., the button's UI); a default implementation of that user interface (e.g., the Basic Button); a Java programming language interface that defines the model of the component (e.g., the Button model); and finally, a default implementation of that model (e.g., the Swing Button model).

For most uses of a given Swing component (e.g., a Button), the programmer doesn't need to know or care about any Swing object other than the first of the five listed above. The programmer simply creates a new instance of the first object of the five, and writes code to interact with it. The others are created automatically based upon the settings on the user's machine. By default, the choice of which user interface and model to use is made in the user's preferences files, where an entire "factory" of user-interfaces and models is specified by the user.

What Swing provides

In order to make it easy to migrate from the user interface classes in the AWT to the new Swing user interface classes, Swing provides a parallel set of user interface classes to those in AWT. Each Swing class that has a parallel in the AWT bears a name that is identical to the AWT name, except that the letter "J" is prepended to it. Otherwise, each parallel user interface object contains a superset of the public methods and variables of the corresponding

AWT class. There are roughly 55 user interface building blocks in Swing, including the common items such as buttons, check boxes, radio buttons, combo boxes, menus, and labels, as well more sophisticated items such as tooltips, tabbed panes, tree views, table views, editable text fields, HTML editors, a Color Chooser, a Money Chooser, etc. Each of these Swing user interface classes fully supports the Pluggable Look and Feel architecture.

Providing Direct Accessibility in Swing

In order to get direct access to a Swing program via a non-mainstream modality (e.g., audio, Braille, etc.), the user would need a factory containing a set of user interface classes installed on their system for each of the Swing classes listed above. Then the user would need to specify that this replacement set be used in the appropriate properties file. Finally, the user would need to run Swing programs that didn't explicitly bar the use of alternate user interface factories.

In setting this up, the user has three options: complete replacement of a mainstream user interface with their alternate one(s); having the mainstream interface and the alternate interface(s) working simultaneously (e.g., visual and audio) through the use of the special multi-plexing user interface factory supplied by Sun; or finally, by choosing on a component-by-component basis which user interface class to use.

UNIX Accessibility (Gnome Accessibility Project)

This section of the training outlines the following:

- Overview of the Gnome Accessibility Project and how it relates the UNIX operating system

GNOME Accessibility Project: The Problem we are trying to solve (www.gnome.org)

The goal of the GNOME Accessibility effort is to ensure that people with disabilities can use the standard GNOME desktop user-environment. There are three sub-pieces to delivering such a desktop:

1. defining what it means to be accessible
2. ensuring that all applications that comprise the GNOME desktop conform to that definition of accessibility

building the assistive technologies that people with disabilities use in order to interact with the GNOME user environment

Breaking Down Barriers with GNOME 2.0 **Paving the Way to a New Generation of Accessible Desktop Solutions** (www.gnome.org)

The growing popularity of the GNOME desktop is driving the need for accessible desktop software for the UNIX and Linux platforms, allowing organizations to provide functional solutions for their employees with disabilities. GNOME 2.0 will help you reach these users by providing the infrastructure you need to quickly build innovative solutions for them. By providing breakthrough technologies that are very easy to use, you can quickly integrate support for accessibility in your applications or assistive technology projects without major architectural changes

The Accessibility Framework integrated into the GNOME architecture delivers several key benefits:

- Simple- built-in framework means no major architectural changes required
- Cross-platform – assistive technologies can run on any GNOME-enabled platform
- Uniform – one standard API across all interface components
- Innovative – an extensible system-wide architecture

Sun Microsystems is leading the way in providing accessibility support in GNOME, leveraging much of the expertise developed for the Java

accessibility project. From this expertise has emerged a solid foundation for allowing developers to write accessible applications from the ground up using the major components of the Accessibility Framework:

- An Accessibility Toolkit Application Programming Interface (ATK API) and associated implementation library integrated with the GTK + 2.0 user interface toolkit that provides built-in accessibility support, enabling developers using GTK + widgets to quickly build accessible applications
- An Assistive Technology Service Provider Interface (AT SPI) for developers to interface technologies such as voice command, text-to-speech, screen readers, and screen magnifiers with GNOME accessible applications on any UNIX, Linux or other GNOME-established platform.

User Interface Guidelines for Supporting Accessibility **(www.gnome.org)**

When designing your application's GUI, there are a number of simple guidelines you should follow to ensure that it can be used by as wide an audience as possible, whether in conjunction with assistive technologies or not. Don't be fooled into thinking that this is just a case of "making your GUI usable by people with disabilities", though, and that you shouldn't bother if you know a disabled person is never going to use your application. Following these guidelines will improve the overall usability of your application for everyone who uses it—including you

User Interface Checklist - The User Interface Guidelines for Supporting Accessibility are summarized in this checklist. When testing an application for accessibility, you should go through each of the items in the list and note whether the application passes or fails each test, or does not apply for that application.

How Accessibility Works in GNOME **(www.gnome.org)**

The Accessibility Toolkit (ATK) describes a set of interfaces that need to be implemented by GUI components to make them accessible. The interfaces are toolkit-independent-- implementations could be written for any widget set, such as GTK, Motif or Qt.

The implementation for the GTK widgets is in a module called GAIL (GNOME Accessibility Implementation Library), which is dynamically loadable at runtime by a GTK application. Once loaded, those parts of your application that use standard GTK widgets will have a basic level of

accessibility, without you having to modify your application at all. (If GAIL is not loaded, GTK widgets will have a default accessibility implementation that essentially returns no information, though it nominally conforms to the ATK API.)

Unlike some accessibility frameworks, GNOME doesn't maintain an off-screen model of the desktop that is then interrogated by assistive technologies (ATs). Instead, all the information required by the ATs is provided directly by the running applications, through a toolkit-independent Service Provider Interface (SPI). The SPI provides a means for UNIX-based ATs, such as screen readers and screen magnifiers, to obtain accessibility information from running applications via the relevant "bridge" (see diagram below).

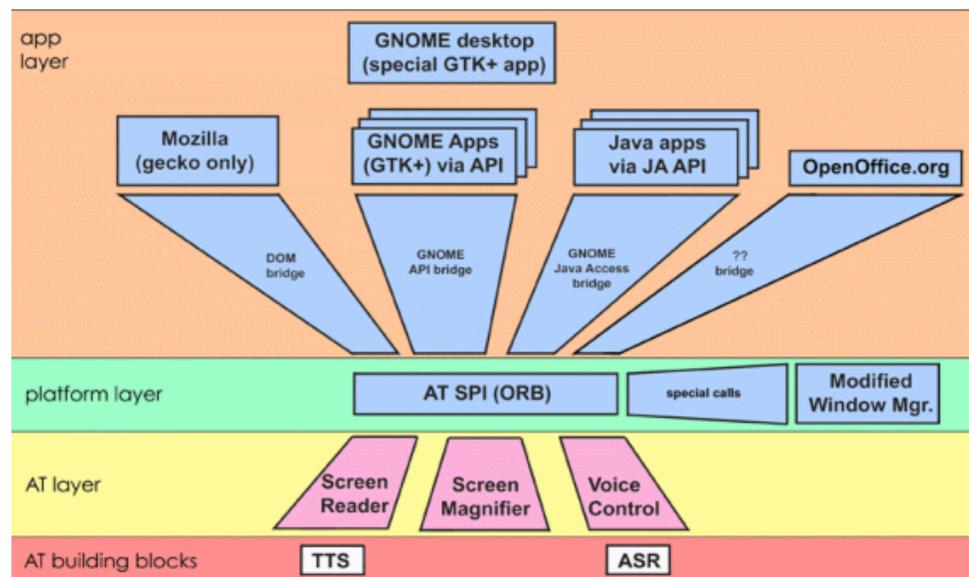


Figure 1. GNOME Accessibility Architecture

You can probably also improve on the default descriptions provided for some of the widgets, and tailor them to that widget's specific purpose in your application. You should add or change the textual descriptions for these widgets with the appropriate ATK function call, so that an assistive technology can describe their purpose or state to the user.

See [Coding Guidelines for Supporting Accessibility](#) for more information.

See [Making Custom Components Accessible](#) and [Examples that Use the Accessibility API](#) for more information.

Microsoft Active Accessibility

This section of the training outlines the following:

- Overview of the Accessibility Features of Microsoft Active Accessibility utilized in Microsoft software applications and operating systems.

Microsoft Active Accessibility (MSAA)

(www.microsoft.com/enable)

Microsoft® Active Accessibility® 2.0 is a set of COM interfaces and APIs that provides a reliable way to expose and collect information about Microsoft Windows-based user interface (UI) elements and Web content. Using this information, Assistive Technology Vendors can represent the UI in alternative formats, such as speech or Braille, and voice command and control applications can remotely manipulate the interface. Active Accessibility relies on Windows® technology and can be used in conjunction only with Windows-based controls and other Windows applications.

Supported Platforms

Active Accessibility 2.0 supports most Microsoft platforms and is included as part of the Windows XP operating system. If you do not know which version of Active Accessibility you are running, see *Which Version of Active Accessibility Is Currently Installed?*

If you do not have the latest version of Active Accessibility, use the Microsoft Active Accessibility 2.0 Redistribution Kit (RDK) to upgrade. This RDK contains all of the core system files needed to incorporate Active Accessibility 2.0 technology into client and server applications. It supports Microsoft Windows® 98, Microsoft Windows NT® version 4.0 with Service Pack 6 or later, and Microsoft Windows XP.

How Active Accessibility Works

Microsoft® Active Accessibility® is designed to help accessibility aids, called *clients*, interact with standard and custom user interface (UI) elements of other applications and the operating system. An Active Accessibility client is any program that uses Active Accessibility to access, identify, or manipulate the UI elements of an application. Clients

include accessibility aids, automated testing tools, and some computer-based training applications.

Using Active Accessibility, a client application can:

- Query for information—for example, about a UI element at a particular location.
- Receive notifications when information changes—for example, when a control becomes grayed or when a text string changes.
- Carry out actions that affect user interface or document contents—for example, click a push button, drop down a menu, and choose a menu command.

The applications that interact with and provide information for clients are called *servers*. A server uses Active Accessibility to provide information about its UI elements to clients. Any control, module, or application that uses Active Accessibility to expose information about its user interface is considered an Active Accessibility server. Servers communicate with clients by sending event notifications (such as calling **NotifyWinEvent**) and responding to client requests for access to UI elements (such as handling WM_GETOBJECT messages sent from OLEACC). Servers expose information through the **IAccessible** interface.

Using Active Accessibility, a server application can:

- Provide information about its custom user interface objects and the contents of its client windows.
- Send notifications when its user interface changes.

For example, to enable a user to select commands verbally from a word processor's custom toolbar, a speech recognition program must have information about that toolbar. The word processor would therefore need to make that information available. Active Accessibility provides the means for the word processor to expose information about its custom toolbar and for the speech recognition program to get that information.

Active Accessibility Basics (www.microsoft.com/enable)

Here are the main feature areas of Microsoft® Active Accessibility®. This includes the following topics:

- Active Accessibility Objects (Only feature covered)
- Types of IAccessible Support
- Client-Server Communication

Active Accessibility Objects

In Active Accessibility terminology, there are *accessible objects* and *simple elements*. Although most applications contain both, accessible objects are more common than simple elements. This section defines and discusses accessible objects and simple elements and provides appropriate UI examples. For more information, see the following topics:

- Accessible Objects
- Simple Elements
- How Child IDs Are Used in Parameters
- Custom User Interface Elements
- Dual Interfaces: IAccessible and IDispatch
- System-Provided User Interface Elements

Accessible Objects

With Microsoft Active Accessibility, user interface (UI) elements are exposed to clients as COM objects. These *accessible objects* maintain pieces of information, called *properties*, which describe the object's name, screen location, and other information needed by accessibility aids. Accessible objects also provide *methods* that clients call to cause the object to perform some action. Accessible objects that have simple elements associated with them are also called *parents*, or *containers*. Accessible objects are implemented using Active Accessibility's COM-based **IAccessible** interface. The **IAccessible** methods and properties enable client applications to get information about UI elements needed by users. For example, **IAccessible::get_accParent** returns an interface pointer to an accessible object's parent, and **IAccessible::accNavigate** provides a means for clients to get information about other objects within a container.

Simple Elements

A *simple element* is a UI element that shares an **IAccessible** object with other elements and relies on that **IAccessible** object (typically its parent) to expose its properties. To differentiate between the elements sharing an **IAccessible** object, the server assigns a unique, positive child identifier to each simple element. This assignment is done on a per-instance-of-interface basis, so the IDs must be unique within that context. Many implementations assign these IDs sequentially, beginning with 1. This scheme does not allow simple elements to have children of their own. Simple elements are also known as *children*.

To be uniquely identified and exposed, a simple element requires an **IAccessible** object and child ID. Therefore, when communicating with

an **IAccessible** object, the clients must supply the appropriate child ID. A special identifier, **CHILDDID_SELF**, can be used to refer to the accessible object itself, instead of one of its children.

The **IAccessible** object shared among simple elements often corresponds to a common parent object in the user interface. For example, system list boxes expose an accessible object for the overall list box and simple elements for each list box item. In this case, the **IAccessible** object for the list box is also the parent or container of the list items.

For more information about accessible objects, see Accessible Objects.

How Child IDs Are Used in Parameters

This topic describes input parameters, output parameters, and special cases for interpreting child IDs returned from **IAccessible** methods.

Input Parameters

Many of the Active Accessibility functions and most of the **IAccessible** properties take a **VARIANT** input parameter. For most of the **IAccessible** properties, this parameter allows client developers to specify whether they want information about the object itself or about one of the object's simple elements.

Active Accessibility provides the constant **CHILDDID_SELF** to indicate that information is needed about the object itself. To obtain information about a simple element, client developers specify its child ID in the **VARIANT** parameter.

When initializing a **VARIANT** parameter, be sure to specify **VT_I4** in the **vt** member in addition to specifying the child ID value (or **CHILDDID_SELF**) in the **IVal** member.

For example, to get the name of an object, and not one of the object's child elements, initialize the variant for the first parameter of **IAccessible::get_accName** (**CHILDDID_SELF** in the **IVal** member and **VT_I4** in the **vt** member), and then call **IAccessible::get_accName**.

Output Parameters

Several **IAccessible** functions and methods have a **VARIANT*** output parameter that contains a child ID or an **IDispatch** interface pointer to a child object. There are different steps that a client must take depending on whether they receive a **VT_I4** child ID (simple element) or an **IDispatch** interface pointer with **CHILDDID_SELF** (full object).

Following these steps will provide an **IAccessible** interface pointer and child ID that together allow clients to use the **IAccessible** methods and properties. These steps apply to the **IAccessible accHitTest**, **get_accFocus**, and **get_accSelection** methods. They also apply to the **AccessibleObjectFromEvent**, **AccessibleObjectFromPoint**, and **AccessibleObjectFromWindow** client functions.

The following table lists the possible result returned and the required

post-processing steps so that clients will have an **IAccessible** interface pointer and child ID.

Result returned

Post-processing for the return value

IDispatch interface pointer

This is a full object.
Call **QueryInterface** to access the **IAccessible** interface pointer.
Use the **IAccessible** interface pointer with **CHILDDID_SELF** to access **IAccessible** methods and properties.

VT_I4 childID

Call **IAccessible::get_accChild** using the childID to see if you have an **IDispatch** interface pointer.
If you get an **IDispatch** interface pointer, use it with **CHILDDID_SELF** to access **IAccessible** interface methods and properties.
If the call to **get_accChild** fails, you have a simple element. Use the original **IAccessible** interface pointer (the one you used in your call to the method or function mentioned above) with the **VT_I4** childID that the call returned.

Before you can use a **VARIANT** parameter, you must initialize it by calling the **VariantInit** COM function. When finished with the structure, call **VariantClear** to free the memory reserved for that **VARIANT**.

Special Cases

There are exceptions to the guidelines in the above table, such as when a child ID is returned by the **IAccessible::accHitTest** method. Servers *must* return an **IDispatch** interface if the child is an accessible object. If a child ID is returned by **IAccessible::accHitTest**, the child is a simple element.

In addition, there are special cases for **accNavigate**. For more information, see **IAccessible::accNavigate** and **Spatial and Logical Navigation**.

Custom User Interface Elements

Server developers design accessible objects based on an application's user interface (UI). Because Active Accessibility implements the

IAccessible interface on behalf of system-provided user interface elements such as list boxes, menus, and trackbar controls, you need to implement the **IAccessible** interface only for the following kinds of custom UI elements:

- Custom controls created by registering an application-defined window class
- Custom controls drawn directly on the screen that do not have an associated HWND
- Custom controls such as Microsoft ActiveX® and Java controls
- Controls or objects in the application's client window that aren't already exposed

Owner-drawn controls and menus are accessible as long as you follow the guidelines discussed in Shortcuts for Exposing Custom User Interface Elements. If you follow these guidelines, then you do not need to implement the **IAccessible** interface for owner-drawn controls and menus.

In most cases, superclassed and subclassed controls are accessible because the system handles the basic functionality of the control. However, if a superclassed or subclassed control significantly modifies the behavior of the system-provided control on which it is based, then you must implement the **IAccessible** interface. For more information, see Exposing Controls Based on System Controls.

If an application uses only system-provided user interface elements, then it does not need to implement **IAccessible**, except for its client window. For example, an application that includes a text editor, not implemented using an edit control, exposes lines of text as accessible objects. Note that Active Accessibility automatically exposes the text in edit and rich edit controls as a single string of text in the **Value** property of the control.

Dual Interfaces: IAccessible and IDispatch

Server developers must provide the standard COM interface **IDispatch** for their accessible objects. The **IDispatch** interface allows client applications written in Microsoft Visual Basic® and various scripting languages to use the methods and properties exposed by **IAccessible**. Since an accessible object provides access to an object either indirectly through **IDispatch::Invoke** or directly with **IAccessible**, it is said to have a dual interface.

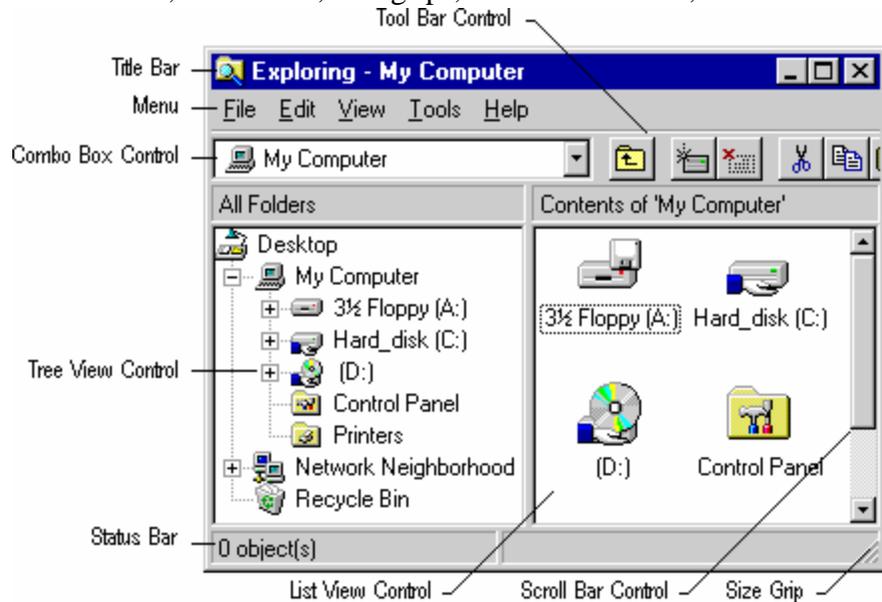
When C/C++ clients get back an **IDispatch** interface pointer, clients can call **QueryInterface** to try converting the **IDispatch** interface pointer to an **IAccessible** interface pointer. To call the **IAccessible** methods indirectly, C/C++ clients call **IDispatch::Invoke**. For improved performance, call the **IAccessible** methods to use the object directly. For a list of the dispatch IDs (DISPIDs) that **IDispatch** uses to identify the **IAccessible** methods and properties, see Appendix C: **IAccessible**

DISPIDs.

System-Provided User Interface Elements

(www.microsoft.com/enable)

Active Accessibility provides support for most predefined and common controls. The following illustration shows a typical window and some of the system-provided user interface elements that Active Accessibility exposes, such as title bars, menus, combo boxes, toolbar controls, tree view controls, status bars, size grips, list view controls, and scroll bars.



Active Accessibility exposes system-provided user interface elements to server applications without requiring the server developer to implement the **IAccessible** interface. Any application that contains these elements automatically inherits their accessibility.

For a list of the controls and other system-provided user interface elements that Active Accessibility supports, see Appendix A: Supported User Interface Elements Reference.

Part 3: Software Applications and Operating Systems Resources

MICROSOFT

[Microsoft Accessibility](http://www.microsoft.com/enable/default.htm): <http://www.microsoft.com/enable/default.htm>

[Microsoft Accessibility for Developers \(MSDN Area\)](http://www.msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000544):

<http://www.msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000544>

ORACLE

[Oracle Accessibility Program](http://www.oracle.com/accessibility.faq.html): <http://www.oracle.com/accessibility.faq.html>

JAVA

[Sun Java Accessibility Site \(text version\)](http://java.sun.com/products/jfc/jaccess-1.3/doc/index.html):

<http://java.sun.com/products/jfc/jaccess-1.3/doc/index.html>

[Sun Java Foundation Classes](http://java.sun.com/products/jfc/): <http://java.sun.com/products/jfc/>

[Sun Accessibility Program](http://www.sun.com/access/general/overview.html): <http://www.sun.com/access/general/overview.html>

[Java Accessibility for Developers](http://www.sun.com/access/developers/index.html): <http://www.sun.com/access/developers/index.html>

GNOME

[GNOME Accessibility for Developers](http://developer.gnome.org/projects/gap/guide/gad/index.html):

<http://developer.gnome.org/projects/gap/guide/gad/index.html>

[Accessibility Design Guidelines](http://developer.gnome.org/projects/gap/hi-design.html):

<http://developer.gnome.org/projects/gap/hi-design.html>

[Assistive Technologies](http://developer.gnome.org/projects/gap/at-types.html):

<http://developer.gnome.org/projects/gap/at-types.html>