IBM® Workplace Forms™

Version 2.6.1

Best Practices for Form Design

S325-2603-00

Note

Before using this information and the product it supports, read the information in "Notices," on page 71.

First Edition (September 2006)

This edition applies to version 1, release 2.6.1 of Workplace Forms and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces version 1, release 2.6 of Workplace Forms.

Contents

Introduction

This document describes *best practices* for designing XFDL forms. These best practices cover a variety of issues, including form security, accessibility, and regulation compliance.

This document divides form design issues into separate sections. Each section contains a basic overview of the design issue as well as a number of best practices for resolving these issues. Each best practice is discussed in the following format:

- **Title of Practice —** Summarizes the practice and provides a brief overview.
- **Why Use This Practice —** Explains why using the practice is important and provides background information for context.
- **Example —** Provides concrete examples that clarify the concepts behind each practice.
- **Usage Notes —** Technical notes about the practice, if necessary.
- **Exceptions To This Practice —** Alerts you to exceptions to the practice that you should consider.

Who Should Read This Document

You should read this document if your organization creates or updates forms that must comply with accessibility and security requirements, withstand legal scrutiny, or has particular certificate requirements. This document is intended for form developers who are familiar with:

- General form design principles
- Digital signatures
- Extensible Forms Description Language (XFDL)
- IBM® Workplace Forms™ Designer
- Digital certificates

For more information about XFDL, refer to the *XFDL Specification*. For more information about the Designer refer to *IBM Workplace Forms Designer Getting Started* guide.

General Best Practices

This section describes general best practices for form design. Practices that involve adding formats to fields, signature security, accessibility, or certificate filters are discussed in later sections.

Topics Discussed

This section discusses the following best practices:

- "Using Four Digit Dates."
- "Do Not Use Smartfill To Store Sensitive Data" on page 4.
- "Localizing Forms" on page 5
- "Do Not Use a Text Editor to Create Your Form" on page 5
- "Do Not Use a Single Field for Entering Large Amounts of Text" on page 6
- "Use SSL Authentication Page" on page 6
- "Do Not Use Progressive JPEG Images" on page 7
- "Do Not Place Input Items On Top Of Labels" on page 7
- "Size Check Boxes and Radio Buttons Using Font Size" on page 7
- "Test Forms Thoroughly" on page 8

Using Four Digit Dates

Using four digit dates allows you to avoid the difficulties associated with using only two digits to represent a year (the Year 2000 problem).

Why Use This Practice

Using four digit dates ensures that your forms and applications are always 'Year 2000' compliant.

The Year 2000 problem occurs when software or hardware handles dates by storing the year in two digits (for example, 99 for 1999), and assumes that the two missing digits are 19. It will display January 6, 1900 and January 6, 2000 the same way (06/01/00) and will interpret both as January 6, 1900.

When handling dates of January 1, 2000 or greater, this becomes an issue. While the Viewer is guaranteed to be Year 2000 compliant (that is, it uses a sliding scale to determine whether two digit dates should be read as 19xx or 20xx), the same is not necessarily true for other applications and third-party technology. To ensure Year 2000 compliance with all of your forms applications, you should ensure that your forms use four digit dates.

Example

The easiest way to ensure that your forms always use four digit dates is to set the *format* option of items that collect date information. The following code sample shows an item with a *format* option with a *datatype* of **date** and a *presentation style* of **long**. This format automatically renders Jan. 31, '03 as **31st January 2003**.

```
<format>
   <datatype>date</datatype>
   <presentation>
      <style>long</style>
   </presentation>
</format>
```

Usage Notes

1. In fields, lists, popups, combo boxes (and sometimes labels) that display a year, always use formats that require and display the year in four digits.
   - Always set a data type of date or year.
   - Do not create a field without a data type.
   - For *year* fields, set a format type of **abbreviated**, **numeric** or **long**, or create a customized formatting template that stores the year as four digits.
   - Do not use short format for dates, and do not create a customized formatting template that converts the year to two digits.
   - For *date* fields, set a format type of **abbreviated**, **numeric** or **long**, or a customized formatting type that stores the year as four digits.
   - Never create customized formatting templates that stores the year as two digits.
2. Always ask your users to enter the year in four digits. Create a label above data entry fields that prompts users to type the date correctly. In the help associated with the field, include instructions to use a four digit year.
3. Always send dates to your database in four digit format.

Exceptions To This Practice

There are no exceptions to this practice.

Do Not Use Smartfill To Store Sensitive Data

Do not use Smartfill data fragments to store sensitive data, such as banking information, credit card information, or personal ID numbers.

Smartfill is a feature that automatically fills out portions of a form in the Viewer. This is accomplished by storing commonly used information, such as the user's name and address, on the user's computer. The Viewer can then access this information at any time, using it to automatically complete sections of forms that require it.

The first time the Viewer opens a Smartfill form, data from the form is saved to the user's computer. Thereafter, each time the Viewer encounters the same Smartfill data, the Viewer will offer to automatically complete that section of the form for the user.

Why Use This Practice

The data fragments used by Smartfill are stored on users' computers, which means that other users may be able to access that information. For this reason, you should never use Smartfill to store sensitive information, such as credit card numbers.

Example

No example provided.

Exceptions To This Practice

There are no exceptions to this practice.

Localizing Forms

You should localize your forms to suit the locale (country and language) in which they will be used.

Why Use This Practice

Every locale uses different symbols to express common concepts, such as currencies, decimal and grouping separators, and mathematical symbols. Furthermore, most locales express dates and times in different ways as well. To ensure that your form displays this information correctly, you need to localize your forms.

Example

To localize your form, you need to configure it to:
- Set the locale
- Select an appropriate font
- Set an appropriate page size
- Correctly format XFDL items
- Create localized signatures

For detailed information on creating localized forms, see the *Locale Specification for XFDL*.

Exceptions To This Practice

There are no exceptions to this practice.

Do Not Use a Text Editor to Create Your Form

Some text editors will automatically add a byte-order mark (BOM) to documents. If you use a text editor to edit a form, configure your editor so it does not add a BOM to UTF-8 encoded documents. (See your text editor’s documentation for details.)

Why Use This Practice

The Viewer and API cannot read UTF-8 encoded forms that have a BOM and will produce the following error: *Unable to load form: Invalid document structure*.

Example

No example provided.

Usage Notes

To remove a BOM from a form, open the form in the Designer and save it.

Exceptions To This Practice

There are no exceptions to this practice.

Do Not Use a Single Field for Entering Large Amounts of Text

Do not require users to enter a large amount of text within a single field. If you expect users to enter several lines of text within a field, inform your users of the maximum number of characters they should enter. If the user input can be naturally divided into components, provide several separate fields for entering each component.

Why Use This Practice

Text fields that the Viewer displays can support up to 500K of text. If the user tries to insert more than 500K of text (about 7,500 lines in a 60-character wide field), the text will be truncated at the 500K mark.

Example

No example provided.

Exceptions To This Practice

There are no exceptions to this practice.

Use SSL Authentication Page

If your users are going to perform submissions using SSL under Internet Explorer 5.0 or 5.5, we recommend that you implement an HTML page that requires SSL authentication, and that users must view before opening a form.

Why Use This Practice

Under Internet Explorer 5.0 and 5.5, the submission of a form using SSL will fail if the browser presents any SSL related dialog box. This is primarily of concern when the browser presents a dialog requesting that the user choose an SSL identity (that is, a certificate). Once the user has selected an SSL identity, subsequent submissions will work properly.

If your users are going to perform submissions using SSL under Internet Explorer 5.0 or 5.5, we recommend that you implement an HTML page that requires SSL authentication, and that users must view before opening a form. This forces the selection of the user’s SSL identity. Once the SSL identity is selected, the browser will not present the selection dialog box again, and SSL submissions will succeed.

Note: Users who protect their certificates with a password may still need to enter that password, but this will not prevent a successful submission.

Example

No example provided.

Exceptions To This Practice

If all of your users are using Internet Explorer 6.0 or higher, you do not need to follow this practice.

Do Not Use Progressive JPEG Images

Do not use Progressive JPEG images in your forms.

Why Use This Practice

The Viewer supports Standard JPEG image formats and does not support Progressive JPEG image formats. If your image is a Progressive JPEG, you will need to open it in a graphics program and save it as a Standard JPEG.

Example

No example provided.

Exceptions To This Practice

There are no exceptions to this practice.

Do Not Place Input Items On Top Of Labels

Avoid placing input items (such as fields, buttons, popups, lists, comboboxes, radios, and check boxes) on top of labels.

Why Use This Practice

When input items are placed on top of labels, the user cannot use the mouse to give the items the focus or to interact with the item (for example, the user cannot click a check box to turn it on or off). The user must use the Tab key to move to the item and the keyboard to enter information in the item.

Example

No example provided.

Exceptions To This Practice

There are no exceptions to this practice.

Size Check Boxes and Radio Buttons Using Font Size

To change the drawn size of a check box or radio button, change the font size of the item.

Why Use This Practice

In the Viewer, the *size* option in check and radio items changes the size of the bounding box around the check box and radio button, not the item's actual drawn size.

Example

No example provided.

Exceptions To This Practice

There are no exceptions to this practice.

Test Forms at Various Zoom Levels

Test your form thoroughly using various zoom levels. If you notice any problems with text or fields, either use a different font or instruct your users to view the form at 100% zoom level.

Why Use This Practice

Some fonts do not scale well when zooming the form, which may cause text to appear cut off in some fields. This problem is most noticeable in fields that are either very small or very large.

Example

No example provided.

Exceptions To This Practice

There are no exceptions to this practice.

Test Forms Thoroughly

You should always test your forms before releasing them for general use. Don't forget to test your forms using all of the accessibility software that a person with disabilities may use. Tests should include:

- Turn on your focus indicator.
- Use various zoom levels.
- Check tab order.
- Verify that all the computes work.
- Use a screen magnifier.
- Review the forms with the three supported screen readers.
- Use keyboard commands only. (Do *not* use a mouse - many users with disabilities cannot navigate with a mouse.)
- Close your eyes or turn *off* your monitor.

Why Use This Practice

Testing your forms ensures that the user experience is easy and intuitive. Tab order should proceed as expected by your users (left-to-right, right-to-left, or top-down depending on the locale), computes should function properly (taking users to correct pages, making items visible or available as required, and so on), and selected fonts should scale properly at all zoom levels so that text is always visible. Testing your forms is especially important for users with disabilities who need to use accessibility tools to access forms. This practice ensures that all visual elements in the form are represented by text that the screen readers can read aloud, that tabbing sequences include all form items and proceed in a logical order, and that the form's background colors clearly contrast with text, mandatory and invalid fields, and the focus indicator.

Example

No example provided.

Exceptions To This Practice

Currently there are no exceptions to this practice.

Formatting Fields

Formatting fields allows you to create datatype, presentation, or constraint rules on data entered into the field. This allows you to add currency symbols to currency fields, pre-format dates, phone numbers or social insurance numbers, zip codes and so on.

This section provides examples for creating some of the most popular field formats. The following topics are discussed:

- "Formatting Phone Numbers."
- "Formatting Postal Codes" on page 12.
- "Formatting E-Mail Addresses" on page 14.

Formatting Phone Numbers

Phone number fields come in a number of formats. Some locales may include periods, others dashes, while others include parentheses. In your form, you may want to accept a wide range of phone number formats, force a particular phone number format, or ensure that users can only input numbers and not letters. You can do this by defining the field's presentation format and providing input constraints.

Input Constraints

If you are creating a form that may have an international audience, it is best to accept a wide range of phone number formats. In particular, you must specify the *patterns* that the phone number field will accept. For example, to specify a three digit area code followed by a seven digit phone number, you would create a pattern that allows:

1. optional parentheses around a three digit area code
2. optional dash, dot, or space
3. 3 digits
4. optional dash, dot, or space
5. 4 digits

You must use a Unix-style regular expression to specify this pattern. For example, the following regular expression creates the pattern described above:

```
(?(\d{3})\)?[[-.][:whitespace:]]?(\d{3})[[-.][:whitespace:]]?(\d{4})
```

A field formatted with this expression would accept user input formatted in many ways, including:

- ##########
- (###)###-####
- (###) ###-####
- ### ### ####
- ###.###.####
- ###-###-####

However, it would reject user input that contained 11 or more numbers or which included alphabetic characters.

For more information about Unix-style regular expressions, see http://www.regular-expressions.info/. For detailed information about formatting fields, see the format option section of the *XFDL Specification*.

Presentation

For every pattern you create as an input constraint, you must have a corresponding presentation *patternref*. The *patternref* allows you to specify how the user's input is displayed in the form. For example, if you want to display the user input as ###-###-#### you need to specify that dashes are placed between each string of digits. You must use a Unix-style regular expression to specify this pattern. For example:

```
$1-$2-$3
```

Examples

The following code sample shows how to accept multiple phone number formats while ensuing that the data is displayed as ###-###-####. For example, 123-456-7890.

```
<field sid="FIELD2">
   <label>Phone Number</label>
      <format>
         <datatype>string</datatype>
         <constraints>
            <patterns>
               <pattern>\(?(\d{3})\)?[[-.][:whitespace:]]?(\d{3})[[-.]&#xA;
               [:whitespace:]]?(\d{4})</pattern>
            </patterns>
         </constraints>
         <presentation>
            <patternrefs>
               <patternref>$1-$2-$3</patternref>
            </patternrefs>
         </presentation>
      </format>
      <value></value>
   </field>
```

Exceptions To This Practice

There are no exceptions to this practice.

Formatting Postal Codes

Postal codes (also known as post codes or ZIP codes) are a series of characters appended to a postal address for the purpose of sorting mail. Most countries use 4, 5, 6, or 9 digit numeric strings, while others use both numbers and letters. In your form, you may want to accept a wide range of valid postal code formats while filtering invalid ones. You can do this by defining the field's presentation format and providing input constraints.

Input Constraints

In particular, you must specify the *patterns* that the postal code fields may accept. For example, to specify a Canadian postal code (two sets of three alternating alphanumeric characters, such as V9A 1G2) you would create a pattern that allows:

- a letter
- a number
- a letter
- an optional space
- a number
- a letter
- a number

You must use a Unix-style regular expression to specify this pattern. For example, the following regular expression creates the pattern described above:

```
([A-Za-z]{1}[0-9]{1}[A-Za-z]{1})\s?([0-9]{1}[A-Za-z]{1}[0-9]{1})
```

A field formatted with only this expression would accept user input formatted in only two ways:

- *x#x #x#*
- *x#x#x#*

To add additional postal code styles, you need to add additional acceptable patterns. For example, to add 4, 5, or 6 character digit strings, or a 9 digit string that may contain a dash, you would add the following patterns:

```
(\d{4})
(\d{5})
(\d{6})
(\d{5})-?(\d{4})
```

For more information about Unix-style regular expressions, see http://www.regular-expressions.info/. For detailed information about formatting fields, see the format option section of the *XFDL Specification*.

Presentation

For every pattern you create as an input constraint, you must have a corresponding presentation *patternref*. The *patternref* allows you to specify how the user's input is displayed in the form. If there are multiple patterns, the first *patternref* corresponds to the first pattern, the second *patternref* corresponds to the second pattern, and so on. You must use a Unix-style regular expression to specify this *patternref*. The follow example shows the *patternrefs* for the 6 character alphanumeric code, the 4, 5, and 6 digit codes, and the 9 digit code that contains a dash:

```
$1 $2
$1
$1
$1
$1-$2
```

Note that the *patternref* for the 4, 5, and 6 digit codes is exactly the same (indicating a single string). However, each must be listed separately to ensure they correspond with the correct input constraint pattern.

Examples

The following code sample shows how to accept multiple postal code formats and allow them to display in the format chosen by the user. This example also uses *casetype* to ensure that any letters are displayed as upper case:

```
<field sid="FIELD4">
   <label>Postal/ZIP code</label>
   <format>
      <datatype>string</datatype>
      <constraints>
         <patterns>
            <pattern>([A-Za-z]{1}[0-9]{1}[A-Za-z]{1})\s?([0-9]{1}[A-Za-z]&#xA;
              {1}[0-9]{1})</pattern>
            <pattern>(\d{4})</pattern>
            <pattern>(\d{5})</pattern>
            <pattern>(\d{6})</pattern>
            <pattern>(\d{5})-?(\d{4})</pattern>
         </patterns>
      </constraints>
      <presentation>
         <patternrefs>
            <patternref>$1 $2</patternref>
            <patternref>$1</patternref>
            <patternref>$1</patternref>
            <patternref>$1</patternref>
            <patternref>$1-$2</patternref>
         </patternrefs>
         <casetype>upper</casetype>
      </presentation>
   </format>
   <value></value>
</field>
```

Exceptions To This Practice

There are no exceptions to this practice.

Formatting E-Mail Addresses

E-mail addresses only come in one format: a string of alphanumeric characters (which may include punctuation symbols), followed by the @ sign, and concluded with a domain name. There are two primary ways to format e-mail address fields:

- To accept any e-mail address
- To accept only e-mail addresses from a particular domain

Accepting Any E-Mail Address

To format a field to accept any e-mail address (and reject any data that is not an e-mail address), you must create an input constraint pattern. You must use a Unix-style regular expression to specify this pattern. For example, the following regular expression configures a field to accept any e-mail address, but rejects any that do not contain an @ symbol or domain name:

```
(?:[A-Za-z0-9]+[._]?){1,}[A-Za-z0-9]+\@(?:(?:[A-Za-z0-9]+[-]?){1,}&#xA;
   [A-Za-z0-9]+\.){1,}[A-Za-z]{2,4}
```

For more information about Unix-style regular expressions, see http://www.regular-expressions.info/. For detailed information about formatting fields, see the format option section of the *XFDL Specification*.

Accepting Only E-Mail Addresses From a Specific Domain

To format a field to only accept e-mail addresses from a particular domain, you must create an input constraint pattern that specifies the required domain name. You must use a Unix-style regular expression to specify this pattern. For example, the following regular expression configures a field to accept only e-mail addresses with an IBM domain:

```
(?:[A-Za-z0-9]+[._]?){1,}[A-Za-z0-9]+\@ibm\.com
```

For more information about Unix-style regular expressions, see http://www.regular-expressions.info/. For detailed information about formatting fields, see the format option section of the *XFDL Specification*.

Examples

The following code sample shows how to create a field that accepts any e-mail address:

```
<field sid="FIELD24">
   <label>E-Mail Address</label>
   <format>
      <datatype>string</datatype>
      <constraints>
         <patterns>
            <pattern>(?:[A-Za-z0-9]+[._]?){1,}[A-Za-z0-9]+\@ &#xA;
                (?:(?:[A-Za-z0-9]+[-]?){1,}[A-Za-z0-9]+\.){1,}[A-Za-z]&#xA;
            {2,4}</pattern>
         </patterns>
      </constraints>
  </format>
  <value></value>
</field>
```

The following code sample shows how to create a field that accepts only e-mail addresses from a particular domain:

```
<field sid="FIELD25">
   <label>E-Mail Address</label>
   <format>
      <datatype>string</datatype>
      <constraints>
         <patterns>
            <pattern>(?:[A-Za-z0-9]+[._]?){1,}[A-Za-z0-9]+\@ibm\.com</pattern>
         </patterns>
      </constraints>
   </format>
   <value></value>
</field>
```

Exceptions To This Practice

There are no exceptions to this practice.

Creating Accessible Forms

With the release of Section 508 of the Rehabilitation Act, the US government has issued a set of regulations describing minimum accessibility standards for information technologies. These regulations have two important goals:

1. To give members of the public who have disabilities equal access to government services.
2. To ensure that persons with disabilities have equal access to employment opportunities with the federal government.

If your organization is impacted by these regulations or have other reasons to adopt accessibility standards, you should review the following topics and incorporate them into your forms design process.

Topics Discussed

This section describes practices you should follow whether you are designing new forms to be accessible or updating existing ones. The emphasis is on issues that apply to XFDL forms that will be read by screen reading software. It includes the following topics:

- "Provide Appropriate Accessibility Messages" on page 18.
- "Put Label Text Into acclabels" on page 25.
- "Use Field Items To Display Text Information" on page 26.
- "Place Graphics Inside Buttons" on page 28.
- "Minimize and Explain the Use of Dynamic Content" on page 30.
- "Reset the Form's Tab Order" on page 30.
- "Identify Row and Column Headings" on page 32.
- "Use Contrasting Page Background Colors" on page 33.
- "Use Wizard-Style Forms" on page 34.
- "Avoid Using Clickwrap Signatures" on page 35.
- "Avoid Using Write-Only Fields" on page 37.
- "Turn Off Smartfill" on page 37.
- "Turn Viewer Help On" on page 38.

System Requirements

The Viewer supports the following screen readers:

- JAWS 5.0
- Microsoft® Narrator
- Microsoft Eyes

While the Viewer is compatible with Internet Explorer, Netscape, and Mozilla browsers, only IE supports Microsoft Narrator and Microsoft Eyes screen readers.

Other Resources

Whether a given XFDL document complies with Section 508 regulations depends almost entirely on its design. While these practices will help you create XFDL forms that integrate with screen readers and magnifiers, you will also have to consider several other accessibility design issues not covered here. This document does not address general accessibility practices for form design. If you need more information on this subject, refer to these online resources:

- **www.w3c.org/WAI/** — The Web Accessibility Initiative web site explores technology, guidelines, tools, education and outreach with the goal of improving access to the Internet.
- **www.microsoft.com/enable/** — Discusses Microsoft's built-in accessibility features.
- **www.access-board.gov/sec508/508standards.htm** — Electronic and information technology accessibility standards published by the US Architectural and Transportation Barriers Compliance Board.
- **www-3.ibm.com/able/** — Product and service information for people with disabilities.
- **ncam.wgbh.org/webaccess/** — Accessibility projects by the National Centre for Accessible Media.

Provide Appropriate Accessibility Messages

To meet the accessibility needs of all the users of your form, you must provide descriptive help messages for most items on the form. However, adding these help messages directly to a form's text leaves the form appearing cluttered and over-complicated to sighted users. Furthermore, most screen readers cannot recognize *label* items and therefore cannot announce their contents.

This problem is solved through the use of *accessibility labels* or *acclabels*. *Acclabels* allow you to add completion instructions and item descriptions to each item. All the supported screen readers recognize and read *acclabels*, but users who do not used screen readers will be unaware of the *acclabel's* presence.

You can assign *acclabels* to any item that is capable of receiving input focus.

Why Use This Practice

Although the supported screen readers recognize most items that appear on a form, the default information they provide is generally insufficient to allow users with visual disabilities to complete a form. As a result, you must provide all the necessary information in the item's accessibility message. In fact, without adequate help information your form may not meet minimum accessibility requirements.

Here are some general practices that will help you create useful messages:

- For interactive items such as fields, lists, buttons, and so on, use verbs that indicate the type of action the user must perform. Some good choices are: type, select, check, and press. Avoid using "enter" because it is too vague.
- Mention whether completion of the item is optional or mandatory. You may choose to only mention this for items that are mandatory.
- In the case of fields that only accept certain types of input, explain the format requirements (such as text, numeric, date, and so on). Be as descriptive as necessary.

- Be consistent in the amount and type of information you provide. For example, if you decide to only indicate which fields are mandatory, do not unexpectedly mention that a certain field is optional.
- Use consistent language for items of the same type. For example, you might decide to use the phrase "Select a choice from the list." as part of the accessibility message for popup lists. In that case, you should use the same phrase for every popup list on the form.
- You should consider the experience level of the users of your form. If you expect users to be fairly new to computers, you may need to provide more detailed instructions. On the other hand, if users are experienced or will be using the same form frequently, you may choose to provide shorter messages.
- If the Help functionality is turned on, the screen readers read an item's help message, so *acclabels* do not need to repeat that content. For details on how to automatically turn on the Viewer's Help functionality, see "Use Field Items To Display Text Information" on page 26.

Example

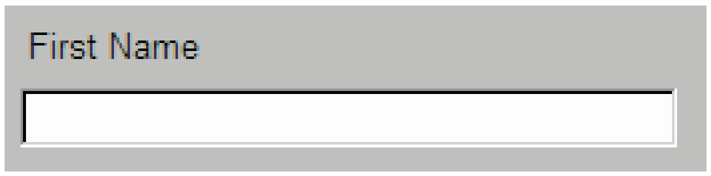
When building an accessible form, it's important to be familiar with the supported screen readers and the default phrases they announce for each item in addition to your custom accessibility message. Before continuing with this section, you may want to review "Appendix A: Screen Reader Announcements" on page 57 for this information.

When creating accessibility messages, try to keep in mind what people with vision disabilities need to know to use an item. The exact wording of each message will depend on the item itself and, to some extent, on the overall design of the form. However, the following examples demonstrate the type of information that you should provide for each item.

Note: In the following examples, messages automatically provided by a screen reader are italicized in the text.

Field

The following diagram shows a typical text field with its associated label.

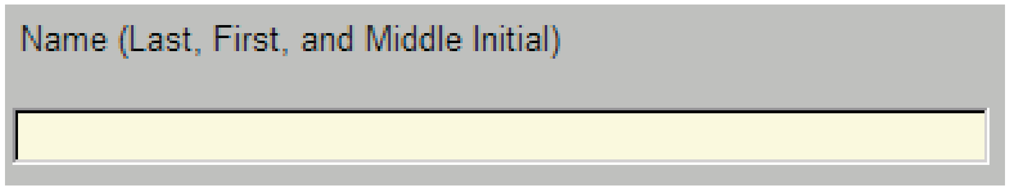


The screen readers announce field messages in the following order:
- label option
- accessibility message
- the value
- screen reader message
- help message
- screen reader tips (JAWS and Narrator only)

Note: For details on screen reader messages and how they integrate with form messages, see "Appendix A: Screen Reader Announcements" on page 57.

Fields often have special formatting requirements. For example, mandatory fields are indicated to sighted users by a yellow shading of the text area. Other special fields may require specific formatting, such as fields that accept only area codes.

The following diagram shows a mandatory text field.



If users with visual disabilities tabbed into this field, they would be unaware that the focus is in a mandatory text field unless you provided an accessibility message for fields. Although screen readers do announce the contents of the field's *label* option, the label does not indicate that the field is mandatory. Additionally, users with cognitive disabilities may require clear instructions to correctly respond to the field.

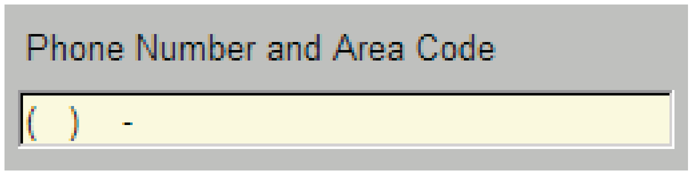
The following code shows a suitable accessibility message for this field:

```
<acclabel>
   Mandatory field. Type your last name, first name, and middle initial.
</acclabel>
```

As a result, when screen reader users move the focus to this field they will hear: "Name left parens last, first, and middle initial right parens. Mandatory field. Type your last name, first name, and middle initial. *Edit field is empty.*"

Note: You should use the verb "type" for fields because this describes the action that users need to perform to complete this item.

The field in the following diagram contains special formatting constraints. This field has been formatted to accept phone numbers and area codes. If users do not enter their area code, they will be informed that their entry is invalid. Note that the field already provides parentheses and a dash - users only need to type the correct numbers.



An appropriate accessibility message for this field could be:

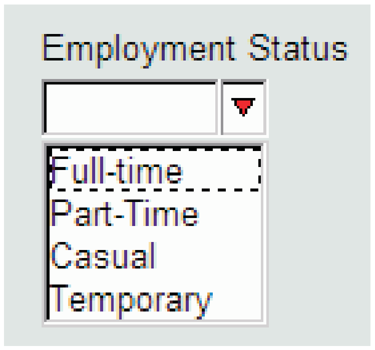
```
<acclabel>
   Type your area code and phone number into this mandatory
   field. You do not need to type the parenthesis around the
   area code or the dash in the phone number. These symbols
   have been provided for you.
</acclabel>
```

When users tab to this field, the screen readers announce: "Phone number. Type your area code and phone number into this mandatory field. You do not need to type the parenthesis around the area code or the dash in the phone number. These symbols have been provided for you. *Edit field is empty.*"

Note: You should note that the supported screen readers pronounce most punctuation, including parentheses, number signs, and dashes that appear in labels, acclabels, and item contents. For example, most screen readers reads the sequence (###) ###-#### as left parens number number number right parens number number number dash number number number number. Not surprisingly, a person with cognitive disabilities could find this confusing. When writing accessibility messages, try to describe formatting requirements in words rather than symbols.

Combobox

Comboboxes allow users to enter text or to select a choice from a list, as illustrated below:



To make comboboxes more accessible to people with visual impairments, you should indicate the total number of choices in the list. The screen readers automatically provide instructions for using the combobox.

The combobox messages are read in the following order:
- label option
- accessibility message
- the selected choice
- screen reader message
- help message
- screen reader tips (JAWS and Narrator only)

Note: For details on screen reader messages and how they integrate with form messages, see "Appendix A: Screen Reader Announcements" on page 57.

Because the screen reader reads the label first, the transition between the accessibility message and the choice announcement may be awkward. At the end of your accessibility message, you may want to repeat the name of the combobox.

The following code demonstrates appropriate code for the *acclabel* of the combobox in the diagram above:

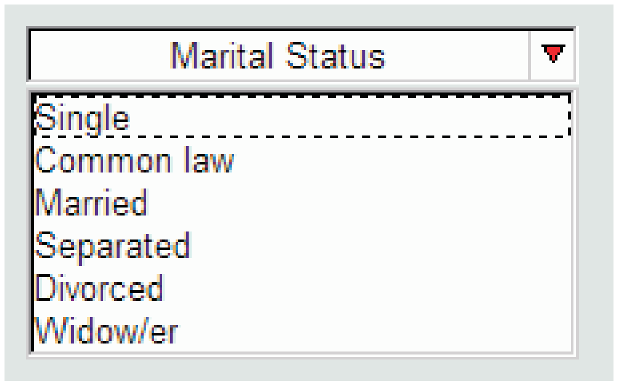
```
<acclabel>
   Type your employment status or select a choice from the list.
   This list contains 4 items. The Employment Status combobox.
</acclabel>
```

When screen reader users move the focus to this field they will hear: "Employment status. Type your employment status or select a choice from the list. This list contains 4 items. The Employment Status combobox *is empty. This is an editable combobox. Type in text or use the down arrow key to choose from the list. Type in text."*

As users move through the list, the screen readers read the choices aloud.

Popup List

Popup lists are the easiest method of offering choices to users without using a lot of space on a form. To make popup lists fully accessible to users with visual impairments, you should indicate the number of choices in the popup. The screen readers automatically announces instructions for using popup lists. The following diagram shows a typical popup list:



Popup messages are read in the following order:
- accessibility message
- label option or selected choice
- screen reader message
- help message
- screen reader tips (JAWS and Narrator only)

Note: For details on screen reader messages and how they integrate with form messages, see "Appendix A: Screen Reader Announcements" on page 57.

Unlike comboboxes, the screen readers say a popup's accessibility message before its label. Therefore, a popup's accessibility message should fully introduce the item, but doesn't need to verify the name of the popup. An appropriate accessibility message would be:

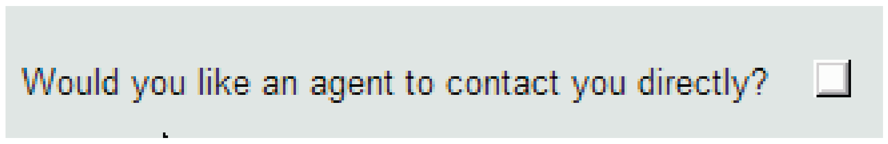
```
<acclabel>
   This popup allows you to select your marital status from a list of 6 choices.
</acclabel>
```

When users access this popup, the screen readers announce: "This popup allows you to select your marital status from a list of 6 choices. Marital Status. *This is a popup list. Use the spacebar or down arrow key to bring up the list. To activate, press spacebar.*"

As users move through the list, the screen readers read the choices aloud.

Check Box

The screen readers read aloud both the check box's accessibility message and label. However, form designers frequently use a separate label item for check boxes to allow more flexible placement options for the label. In that case, to make check boxes accessible, you should repeat the contents of the label in the *acclabel* item. The following diagram shows a check box and its label:



Check box messages are read in the following order:
- label option
- accessibility message
- value
- screen reader message
- help message
- screen reader tips (JAWS and Narrator only)

Note: For details on screen reader messages and how they integrate with form messages, see "Appendix A: Screen Reader Announcements" on page 57.

As always, the accessibility message should indicate what type of action the user needs to perform. The following code shows an appropriate help message for this check box:

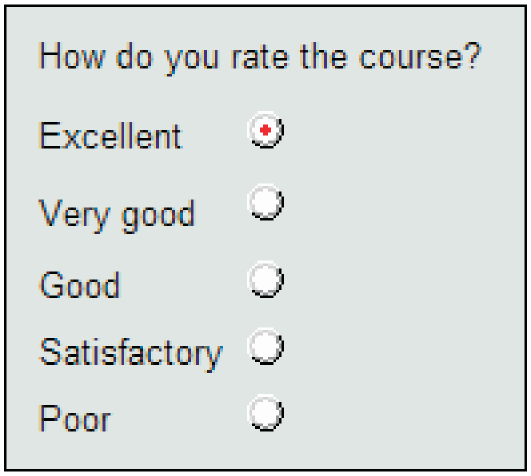
```
<acclabel>
   Select this check box if you would like an agent to contact
   you directly.
</acclabel>
```

Although the message consists of only one sentence, it conveys what action the user can take as well as the question presented by the label. When users access this check box, they hear: "Would you like an agent to contact you directly? Select this check box if you would like an agent to contact you directly. *Check box: checked/not checked. To activate, press spacebar*."

Radio Button

As with check boxes, form designers frequently use separate label items for radio buttons. If this is the case, you must provide this information individually for each radio button by using its accessibility message.

In addition, grouped radio buttons often have a title or caption that applies to the entire group, as shown in the following diagram:



Radio button messages are read in the following order:

- label option
- accessibility message
- value
- screen reader message
- help message
- screen reader tips (JAWS and Narrator only)

Note: For details on screen reader messages and how they integrate with form messages, see "Appendix A: Screen Reader Announcements" on page 57.

Grouped radio buttons typically have a separate label that describes the set of buttons, as illustrated in the diagram above. You should present this information in either a read-only field or in the acclabel of the first radio button, so that it is available to all your users. For details on presenting this information, see "Use Field Items To Display Text Information" on page 26 or "Put Label Text Into acclabels" on page 25.

In addition to containing the text that appears on the form, the accessibility message of the read-only field should indicate that the user is expected to respond using radio buttons. The following code shows an appropriate message for the text field shown above:

```
<acclabel>
   What did you think of the course? Respond by using the
   following radio buttons.
</acclabel>
```

If you prefer not to use read-only fields, you should ensure that this information is placed in the acclabel of the first radio button.

In either case, you must also create individual accessibility message for each radio button. For example, the following code shows an acclabel for the ″Excellent″ radio button:

```
<acclabel>Excellent</acclabel>
```

When users accessed this radio button, they would hear: "What did you think of the course? Respond by using the following radio buttons. Excellent. *Radio button. x of n. Checked/Not checked. To activate, press spacebar*."

Exceptions To This Practice

There are no exceptions to this practice.

Put Label Text Into acclabels

Most forms contain a certain amount of read-only text such as titles, captions, headings, or instructions. This text is normally displayed using label items. You should include this text in the *acclabel* of the appropriate input items.

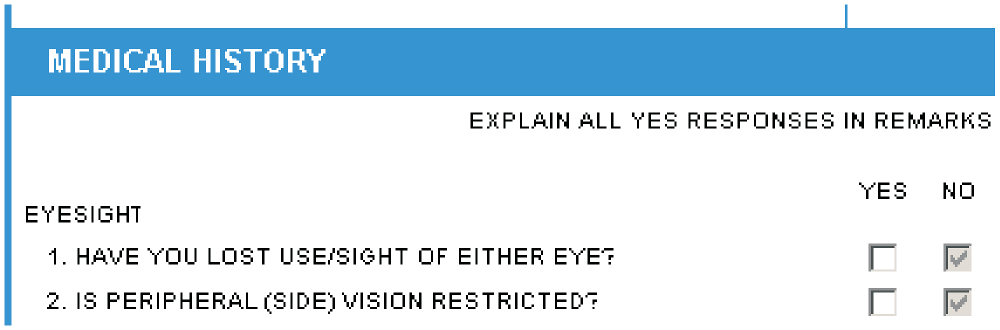
Note: This practice is an alternative to “Use Field Items To Display Text Information” on page 26.

Why Use This Practice

Screen readers only announces text from items that receive the input focus. Because label items are not designed to accept input from the user, they never receive the focus. As a result, the screen reader cannot read a label’s text. If you use labels to separate contextual areas on a form, users with visual disabilities may be unaware of the subject change. If you repeat and expand upon the label information in the accessibility message of the appropriate items, the screen readers can inform users of any changes or instructions.

Example

There are a number of situations in which you would use read-only text information in a form. The following diagram illustrates a section header, followed by instructions for completing medical information:



MEDICAL HISTORY

EXPLAIN ALL YES RESPONSES IN REMARKS

	YES	NO
EYESIGHT		
1. HAVE YOU LOST USE/SIGHT OF EITHER EYE?	☐	✓
2. IS PERIPHERAL (SIDE) VISION RESTRICTED?	☐	✓

The following code creates an *acclabel* option for the "Yes" check box shown in the previous diagram. Note that the check box *acclabel* contains all the information displayed in the section’s read-only labels, including section title, instructions, sub-heading, and the first question of the section:

```
<check sid="LostSight_Yes>
   <acclabel>
      This section of the form records your Medical
      History. It contains a series of questions with yes and no
      check boxes. If you answer yes to any of these questions,
      please explain your response in the Remarks section that
      directly follows the list of questions. Eyesight.
      Question 1. Have you lost the use or sight of either eye?
      Select yes or no. This is the Yes check box.
   </acclabel>
</check>
```

When a screen reader user tabs to this item, they hear: "This section of the form records your Medical History. It contains a series of questions with yes and no check boxes. If you answer yes to any of these questions, please explain your response in the Remarks section that directly follows the list of questions. Eyesight. Question 1. Have you lost the use or sight of either eye? Select yes or no. This is the Yes check box. *Checkbox: not checked. To activate, press spacebar.*"

Usage Notes

When an item receives the focus (for example, by using the tab key to navigate to the field), screen readers typically announce:

- The *label option* of the item.
- The accessibility message.
- The contents of the item.
- Any instructions the screen reader automatically adds.

As a result, you should ensure that the item's *label option* and accessibility message do not repeat information. Repeating, conflicting, or out-of-order messaging can be confusing. If you have items that contain the text of a header, caption, or instruction label in its accessibility message, do not use a *label option* to display labels for those items. Instead, use a separate label item to provide text for sighted users.

Whether you are using the Designer or a text editor to create or modify your form, remember:

- For sighted users, create a *label item* instead of a *label option*.
- For users with visual disabilities, ensure that you place all relevant information (section header, instructions, item information, and so on) in the acclabel.

  Note: This practice may result in lengthy accessibility messages, which is an issue for screen readers that limit message length to 256 characters. This issue can be avoided by splitting instructions across the appropriate items. Where a sighted user might prefer to read all of the instructions at the beginning of a section, it is more useful to split the instructions across multiple items for a vision impaired user. For more information on this issue, see "Use Wizard-Style Forms" on page 34.

Exceptions To This Practice

If the majority of your intended users have vision impairments, you may choose to substitute read-only fields for labels containing section headers, instruction, captions, and so on. For more information on substituting read-only fields, see "Use Field Items To Display Text Information."

If your accessibility messages need to be longer than 256 characters, you may need to re-design your form so that it is simpler to use. For more information, see "Use Wizard-Style Forms" on page 34.

Use Field Items To Display Text Information

As an alternative to putting label text in acclabels, you may choose to avoid the use of *label* items to display text on your forms. Instead, you can use specially formatted *field* items to display label text.

This practice involves creating read-only fields that receive the focus. This is fine if the majority of your users have vision impairments. However, keep in mind that this can reduce the usability of the form for sighted users.

Note: This practice is an alternative to "Put Label Text Into acclabels" on page 25.

Why Use This Practice

Screen readers only read text from items that receive the input focus. Because label items are not designed to accept input from the user, they never receive the focus. As a result, the screen reader cannot read a label's text. For example, if you use labels to give text instructions for completing a form, users with visual disabilities will be unaware of those instructions. By using fields instead of labels, you are ensuring that the information is available to all users. When properly formatted, such fields are virtually indistinguishable from the labels that they replace and will not affect the overall layout of your form.

Note: If sighted users tab through a form formatted this way, they will find that the tab order includes items that appear to be labels, and that a cursor appears when that item has the focus. This may reduce the usability of the form for sighted users.

Example

When a field receives the focus (for example, by using the tab key to navigate to the field), the screen reader first announces the accessibility message for the field, followed by its contents. The following diagram shows part of a form in which a *field* item creates a section heading and gives instructions to the user:

In this case, when screen reader users tab into a read-only field, they hear the reader announce the contents of the field, general field information, and any additional *acclabel* the field may have.

Usage Notes

The ability to provide extra information through an item's accessibility message is an important part of making a form accessible. For more information on providing accessibility messages, refer to page "Provide Appropriate Accessibility Messages" on page 18.

When you replace a regular label with a field, remember to:
- *Not* specify a *label* option for the field.
- Set the field as read-only.
- Set the field as active.
- Disable the border around the field's contents.
- Specify a custom *acclabel* item for the field. As a minimum, the accessibility message should repeat the text that appears on the form. However, it is usually desirable to make the audible message more descriptive than the text it supports.

The following code creates an *acclabel* item for the “Select Items for Purchase” heading shown in the previous diagram:

```
<acclabel>
   This section of the form allows you to select the type and
   amount of the items you want to purchase. The price is
   automatically calculated for you.
</acclabel>
```

When this item receives the input focus, screen reader users hear: “This section of the form allows you to select the type and amount of the items you want to purchase. The price is automatically calculated for you. *Edit field contains Select Items for Purchase. Read-only.*”

Note: If you use labels to identify other items, such as fields, check boxes, or radio buttons, you may not need to exchange read-only fields for labels, as these items can contain their own accessibility messages that describes their function. As a rule of thumb, simply ensure that the screen readers pronounces all visible text on the form whether it be through read-only fields or accessibility messages.

Exceptions To This Practice

Sighted users may find it distracting to see the cursor tab into items that do not require user input. As a result, you may prefer to put label text into the *acclabel* of the first item in the section. For more information, see “Put Label Text Into acclabels” on page 25.

Place Graphics Inside Buttons

If you want to include graphics on your form, you should place each image inside a button that has its border turned off.

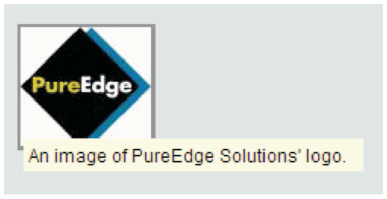
Why Use This Practice

Section 508 regulations require that there be a text equivalent description of non-text elements such as images and graphics. The best way to meet this requirement is to place images and graphics within buttons. This allows you to place a text description of the image in the button’s accessibility and help messages. Removing the button’s border maintains the image’s appearance for sighted users.

When users access the button, the screen readers say the accessibility message aloud. If Viewer Help is turned on when users tab into the button, the readers announce the help message while the form displays a hover help. As a result, the layout of the form is not affected.

Example

The following diagram shows part of a form containing an image of the PureEdge logo. Because the image is contained within a button, the form designer was able to use *help* and *acclabel* items to display descriptive text about the image.



The following code creates the image button shown above:

```
<button sid="BUTTON1">
   <itemlocation>
   <value>BUTTON1</value>
   <image>DATA1</image>
   <borderwidth>0</borderwidth>
   <acclabel>An image of the PureEdge logo</acclabel>
   <help>HELP5</help>
   <fontcolor>
   <bgcolor>
</button>
```

This code creates the help message for the button:

```
<HELP sid="HELP5">
   <value>An image of the PureEdge logo.</value>
</HELP>
```

Note: Remember, Viewer Help must be turned on before the screen readers can read the help messages.

Usage Notes

When using buttons to display images you should remember to:

- Specify an invisible border for the button.
- Make the button the same size as the image.
- Include the image button in your tab layout.

You may be tempted to make the button inactive so that users cannot click it. However, keep in mind that inactive buttons do not receive the focus, and that users with visions impairments must be able to tab to an item to get an accessibility message for it.

Note: Remember that help messages are displayed in text on the screen. Information conveyed in a help message should be applicable for both sighted users and users with visual disabilities.

Exceptions To This Practice

According to Section 508 regulations, you only need to provide text equivalent descriptions for non-text elements that ″provide information required for comprehension of content or to facilitate navigation″. In other words, you do not need to provide text for graphic elements such as lines, frames, and boxes.

Minimize and Explain the Use of Dynamic Content

Dynamic content usually consists of XFDL items whose appearance, operation, or value change at runtime in response to user events involving some other form element. If you are creating or modifying forms that will be used by people with special accessibility needs, you should only include dynamic content if it is essential to the operation of the form. This is particularly true if the dynamic items significantly affect the layout, appearance, or operation of the rest of the form.

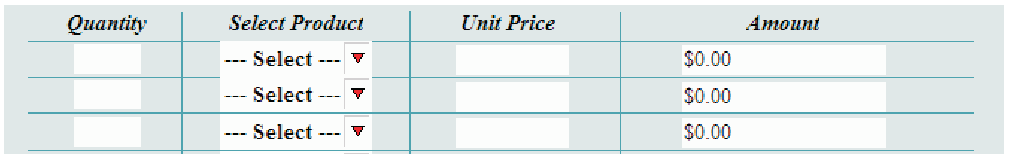
If you find that you must include dynamic content, you should make it as simple as possible. In addition, you should make every effort to alert users of how the form changes and which items are affected. The example section below demonstrates one way of doing this using *acclabel* items.

Why Use This Practice

Depending on its complexity, dynamic content can have a considerable effect on a form's appearance and operation. Although dynamic content is often included to make forms easier to use, persons with visual or cognitive disabilities may not always be aware of these changes and may find the form difficult to understand, or may completely miss changes that affect the overall meaning of the form.

Example

The following diagram shows part of a form containing some simple dynamic content. When users complete the ″Quantity″ and ″Product″ items, the form automatically fills in the ″Unit Price″ and ″Amount″ fields.



Quantity	*Select Product*	*Unit Price*	*Amount*
	--- Select ---		$0.00
	--- Select ---		$0.00
	--- Select ---		$0.00

Although this is a simple example, the form should still identify which fields it updates automatically. The easiest way to do this is to include this information in the affected item's accessibility message. For example, the following code creates the accessibility message for the first ″Amount″ field:

```
<acclabel>
   Amount Column. Row 1. The form automatically calculates this
   amount.
</acclabel>
```

Exceptions To This Practice

Currently there are no exceptions to this practice.

Reset the Form's Tab Order

This practice is a reminder to carefully check and update your form's tab order.

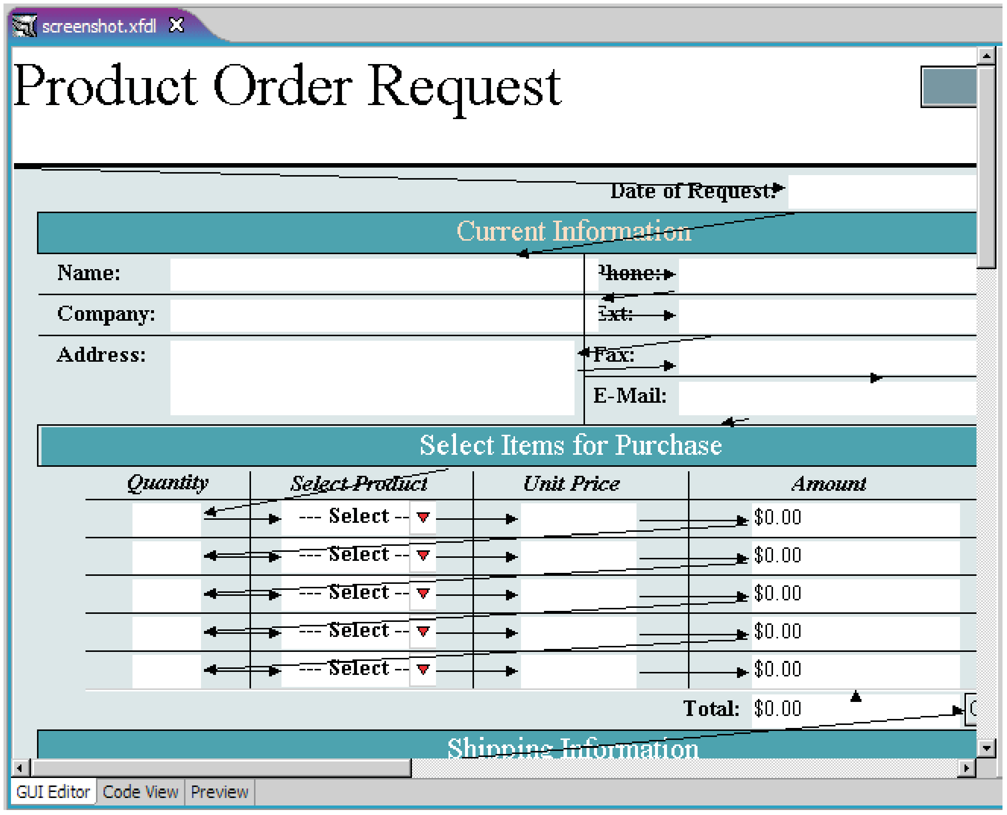
Why Use This Practice

If you modified an existing form to meet Section 508 requirements, you most likely added a number of items to your form. In that case, you must reset the form's tab order so that the focus moves from item to item in a logical order. Keep in mind that many users with disabilities rely solely on keyboard navigation to review and complete forms.

In particular, you should include in the tab order any read-only fields that implement informative text elements such as headings, titles, captions, and instructions. If the form contains a *toolbar* item, you should also include the items that appear in the toolbar.

Example

The following diagram shows a portion of a form in Designer, with the tab order view enabled. The arrows indicate the tab order.



Note that the title of the form (Product Order Request) is located in a toolbar and is the first item in the tab order. To include this title in the tab order and make its accessibility message available to the screen reader, it was implemented using a read-only *field* item, rather than a label. Other fields that implement read-only text include the "Contact Information" and "Select Items for Purchase" headings. These items are also part of the tab order so that screen reader can read their accessibility messages.

Exceptions To This Practice

Currently there are no exceptions to this practice.

Identify Row and Column Headings

If your form contains items arranged in a table layout, you must identify headings for each row and column. This involves placing read-only fields with appropriate accessibility messages at the start of every data column and row.

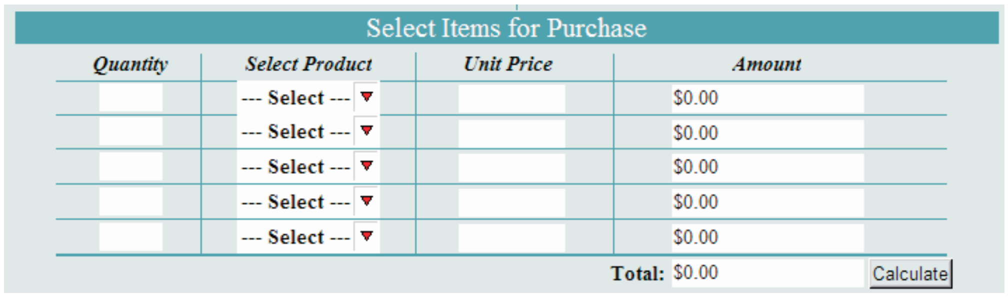
Why Use This Practice

Section 508 regulations require that row and data columns be identified for data tables. The goal of this requirement is to ensure that users of assistive technologies such as JAWS or Microsoft Narrator can correctly interpret tables.

Although XFDL does not support a true table item, it is easy to arrange individual fields and other items in a grid-like pattern, thereby replicating the functionality of a table. In such cases, you should provide row and column headings with accessibility messages that the screen readers can read aloud. You should also include similar accessibility information for each cell, so that users always know their current position within the table.

Example

The following diagram shows a table that enables users to select items for purchase. The table consists of four columns and five rows.



Select Items for Purchase

Quantity	*Select Product*	*Unit Price*	*Amount*
	--- Select ---		$0.00
	--- Select ---		$0.00
	--- Select ---		$0.00
	--- Select ---		$0.00
	--- Select ---		$0.00
		Total:	$0.00 Calculate

Note that every row and column is identified by a unique heading. Each heading consists of a read-only field and an accessibility message. The accessibility message should identify the item as a heading and whether it is a column or row. It is also helpful to number each column or row. The following code shows the accessibility message for the ″Unit Price″ column heading:

```
<acclabel>Column Heading 3 of 4</acclabel>
```

When the focus is on the ″Unit Price″ heading, the screen readers announce ″Column heading 3 of 4. Editable text. Unit Price″.

To help users with visual disabilities be aware of their current position within the table, you should include the column name and row number in the *acclabel* item for each cell. For example, the following code creates a accessibility message for the cell in the third row of the first column:

```
<acclabel>
   Quantity Column. Row 3. Type the quantity of the
   product you would like to order.
</acclabel>
```

Exceptions To This Practice

Currently there are no exceptions to this practice.

Use Contrasting Page Background Colors

You should always use clearly contrasting colors for your text and bgcolors.This makes your form easier to read and understand. Additionally, you should make sure your text and background colors are clearly different than the colors the Viewer uses to display the focus indicator and highlight mandatory and invalid fields. By default, the Viewer shades mandatory fields in yellow and invalid fields in red. The focus indicator is always black. The following list shows the RGB triplet for these reserved colors:

″Mandatory″ Yellow	255 255 208
″Invalid″ Red	255 128 128
″Focus Indicator″ Black	0 0 0

You should either use a page background color that provides adequate contrast from these colors, or choose new contrasting colors to indicate the mandatory or invalid status of items.

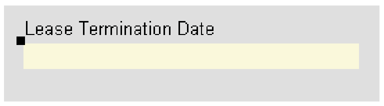
Why Use This Practice

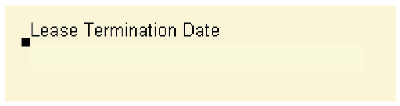
If the enhanced focus indicator is on, the Viewer displays it as a black square. It indicates that the item has the cursor. The text entry area of mandatory fields has a light yellow background, while invalid fields are red. If the background of the form is a similar color, these fields may become difficult to see for people with certain vision disabilities.

While you cannot change the color or symbol used by the focus indicator, it is possible to modify the default colors of the mandatory and invalid fields. For example, some form designers prefer to use a brighter color to indicate mandatory fields. By increasing the contrast of mandatory fields, the designers ensure that the mandatory fields are readily visible to all sighted users.

Example

The following diagrams show part of the same form, but with different page background colors. Note how the text entry area of the mandatory field seems to disappear when item borders are off and the page's background color is set to Lemon Chiffon (255 250 205).





Usage Notes

You can make changes to the default mandatory and invalid colors by adding *ufv_settings* to the form global:

```
<ufv_settings>
   <mandatorycolor>255,165,0</mandatorycolor>
   <errorcolor>HotPink</errorcolor>
</ufv_settings>
```

Exceptions To This Practice

As of Viewer 6.2, users with visual disabilities can choose to override form color settings with high contrast operating system colors. If this option is used, the Viewer automatically keeps a record of those colors so that the appearance of the document, when signed, can be recreated.

Use Wizard-Style Forms

You should use wizard-style forms whenever possible. Wizard-style forms are very simple and easy to use. Unlike forms based on traditional paper forms, which typically feature pages covered with densely packed labels and fields, wizard-style forms feature only one or two related questions per page. Users navigate sequentially through the form using 'Next Page' buttons.

You can use these page flips to dynamically update the form behind the scenes by using user information to determine what pages should be shown next, or to pre-populate other areas of the form with user data. For example, you might design a form in which one section changes completely depending on whether the user is single or married. On the first page of the form the user might indicate whether they are married or single. Then the user clicks a button to flip to the next page. The content of the next page is based purely on the user's marital status. Married users see page two, in which they must provide some information about their spouse and dependents, while single users skip page two entirely and are shown page three.

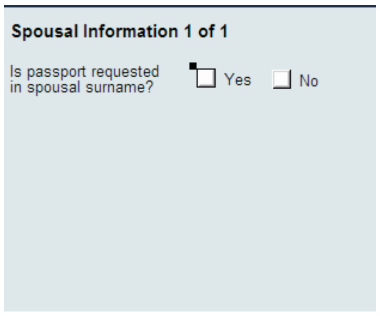
Why Use This Practice

Forms with complex page content can be very difficult for users with visual disabilities to navigate and complete. While using descriptive *acclabels* helps to mitigate these issues, items in complex forms may require longer explanations. Providing complete explanations within the 256 character limit that the screen readers impose on accessibility messages can be challenging. Moving from a traditional form layout to a wizard-style layout minimizes these complexities and helps to reduce the amount of required accessibility messaging.

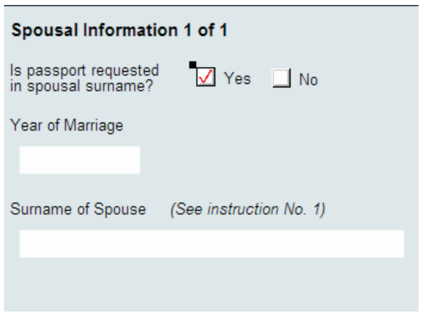
Furthermore, dynamically manipulating a traditional-style form's presentation can be very confusing for someone who cannot see the changes to a form. For example, visually impaired users may not notice that mandatory rows have been added to a table, but they would not be permitted to sign the form until the new row has been completed. In a wizard-style form, the information required by that unexpected table row could simply be presented as a new form page in a series of form pages.

Example

The following pictures illustrate a page in which the spousal information dynamically alters when the user selects the appropriate check box:


Spousal Information 1 of 1

Is passport requested in spousal surname? ☐ Yes ☐ No

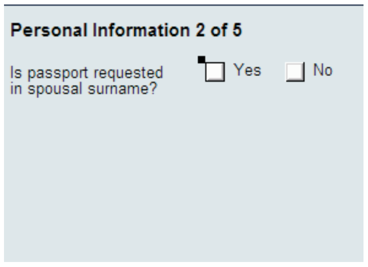

Spousal Information 1 of 1

Is passport requested in spousal surname? ☑ Yes ☐ No

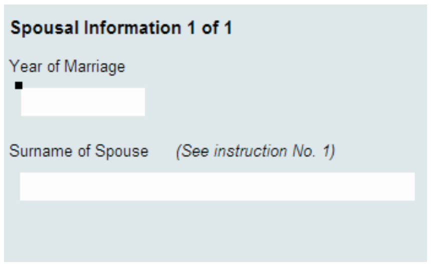
Year of Marriage

Surname of Spouse (See instruction No. 1)

In a more complex form, these new fields could easily be missed by users with visual disabilities. However, in a wizard-style form, you can add a page flip to the first page, as shown:


Personal Information 2 of 5

Is passport requested in spousal surname? ☐ Yes ☐ No


Spousal Information 1 of 1

Year of Marriage

Surname of Spouse (See instruction No. 1)

In this case, the user clicks the **Next** button to move to the next page. This brings up the special ″Spousal Information″ page because the user selected **Yes** in the previous page. This makes the dynamically added fields part of the normal flow of completing the form and provides the user with a more intuitive experience.

Exceptions to this Practice

There are no exceptions to this practice.

Avoid Using Clickwrap Signatures

If you require your users to sign your forms, ensure that you use digital signatures instead of Clickwrap signatures. Digital signatures use unique digital codes to identify a signer and authenticate the document’s data. This allows for a very simple signing ceremony. Clickwrap signatures, on the other hand, do not use encryption or digital certificates for signing. The Clickwrap signing ceremony typically identifies signers through a series of questions and answers and may require users to ’echo’ statements to indicate their agreement with the terms of the document. As a result, the process of completing a Clickwrap signing ceremony is more complicated than signing a form with a digital signature. Furthermore, none of the supported screen reader are capable of reading the entire Clickwrap signing ceremony.

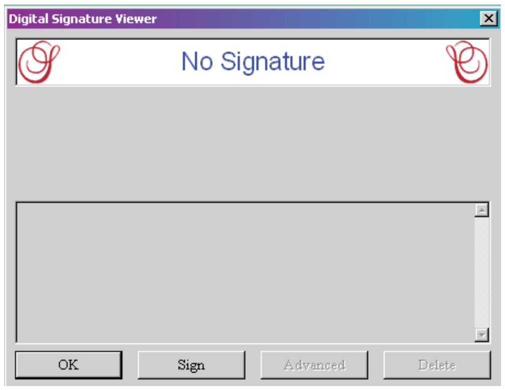
Why Use This Practice

None of the three supported screen readers are able to fully read the Clickwrap signature dialog box. That means valuable signing information is not passed on to users with visual impairments. This lack of information can make the signing ceremony confusing for vision impaired users or prevent them from correctly completing the signing ceremony.

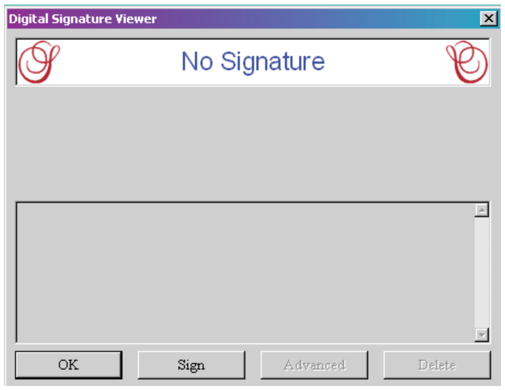
Example

The following examples compare a typical Clickwrap signing ceremony with a digital signature signing ceremony:

Digital Signature Signing Ceremony



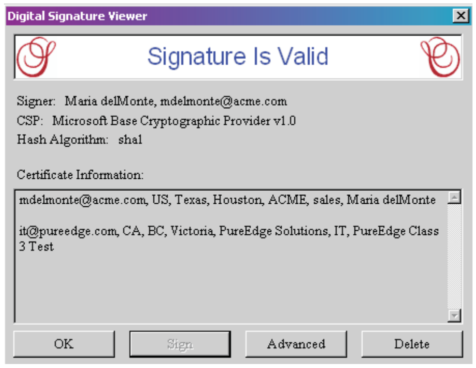
Clickwrap Signature Signing Ceremony



As the signing ceremonies open, they appear identical. Signers are required to simply click the **Sign** button. At this point, however, the content of the ceremonies begin to diverge.

In the case of the digital signature signing ceremony, the ceremony is complete once the **Sign** button has been clicked. But the Clickwrap signing ceremony has only just begun. When the **Sign** button is clicked, users are presented with questions designed to identify the user and text intended to ensure that the user is fully informed and accepts the terms of the agreement:

Digital Signature Signing Ceremony



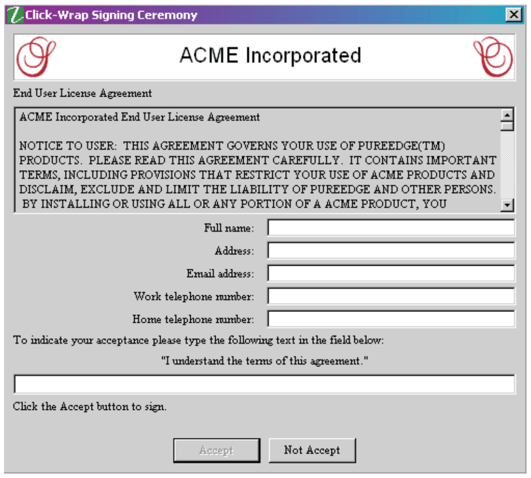
Signer: Maria delMonte, mdelmonte@acme.com
CSP: Microsoft Base Cryptographic Provider v1.0
Hash Algorithm: sha1

Certificate Information:

mdelmonte@acme.com, US, Texas, Houston, ACME, sales, Maria delMonte

it@pureedge.com, CA, BC, Victoria, PureEdge Solutions, IT, PureEdge Class 3 Test

Clickwrap Signature Signing Ceremony



End User License Agreement

ACME Incorporated End User License Agreement

NOTICE TO USER: THIS AGREEMENT GOVERNS YOUR USE OF PUREEDGE(TM) PRODUCTS. PLEASE READ THIS AGREEMENT CAREFULLY. IT CONTAINS IMPORTANT TERMS, INCLUDING PROVISIONS THAT RESTRICT YOUR USE OF ACME PRODUCTS AND DISCLAIM, EXCLUDE AND LIMIT THE LIABILITY OF PUREEDGE AND OTHER PERSONS. BY INSTALLING OR USING ALL OR ANY PORTION OF A ACME PRODUCT, YOU

Full name:

Address:

Email address:

Work telephone number:

Home telephone number:

To indicate your acceptance please type the following text in the field below:

"I understand the terms of this agreement."

Click the Accept button to sign.

As you can see, the Clickwrap signing ceremony is more complex than the digital signature signing ceremony. Given that the screen readers cannot read all of the text in the Clickwrap signing ceremony, this additional complexity could make it impossible for users with vision disabilities to sign the form.

Exceptions To This Practice

Currently there are no exceptions to this practice.

Avoid Using Write-Only Fields

You should avoid using write-only fields in your forms as much as possible. Write-only fields do not display the text typed by the user. Instead, these fields replace each character with an asterisk. They are frequently used to allow users to enter sensitive information, such as passwords. However, some screen readers will read the true text of these fields instead of the replacement characters.

Why Use This Practice

Because some screen readers read the true text contained within write-only fields, sensitive information such as passwords can be obtained by users with screen readers. To protect the privacy of your users, you should avoid using write-only fields wherever possible.

Example

No example provided.

Exceptions To This Practice

If your forms require a write-only field to allow users to enter sensitive information, you should consider creating a compute that turns the write-only field invisible once it has been filled in. Screen readers ignore invisible fields and therefore cannot inadvertently reveal data contained in an invisible field.

Furthermore, you should advise your screen reader users to wear headphones while completing forms containing sensitive information. This ensures that personal information cannot be overheard when it is read aloud by the screen reader.

Turn Off Smartfill

You should advise your users with vision disabilities to turn off Smartfill in their Viewer preferences.

Smartfill is a feature that automatically fills out portions of a form in the Viewer. This is accomplished by storing commonly used information, such as the user’s name and address, on the user’s computer. The Viewer can then access this information at any time, using it to automatically complete sections of forms that require it.

The first time the Viewer opens a Smartfill form, data from the form is saved to the user’s computer. Thereafter, each time the Viewer encounters the same Smartfill data, the Viewer will offer to automatically complete that section of the form for the user.

Why Use This Practice

Unfortunately, none of the three supported screen readers are able to fully read the Smartfill dialog boxes. Therefore, users with vision difficulties will be unaware what data is being provided by Smartfill. This could lead users to accidentally provide information that they do not want to share. In some cases, the lack of support from the screen readers may cause the users to be unable to proceed beyond the Smartfill dialog box to the form itself.

Example

No example provided.

Usage Notes

To turn off Smartfill:

1. Click the **Preferences** button in the Viewer toolbar or press **ALT + F12**.
   - The Preferences dialog box appears.
2. Click **Input Options**.
3. Under **Smartfill**, deselect **Enable Smartfill**.
4. Click **Save**.

Exceptions To This Practice

There are no exceptions to this practice.

Turn Viewer Help On

You can set a form to automatically open with the Viewer's Help mode turned on. This means that help messages are automatically displayed and read aloud to the user. However, keep in mind that this can reduce usability for sighted users, and that this option is best used if you are certain your users have vision impairments.

Help messages give additional instructions for completing items on the form. They may provide special formatting instructions, or simply offer examples of the expected response. Help messages may be particularly useful to people using screen magnifiers or those with cognitive disabilities.

Why Use This Practice

If Viewer help is not turned on, the screen readers cannot read the relevant help messages. Many users with visual disabilities cannot access the toolbar icon that turns on Viewer Help, and may be unaware of the shortcut key that turns it on. This means that unless Viewer Help turns on automatically, many of your users may be unable to access the help messages they need to complete the form.

Example

To automatically turn on Viewer Help when a form opens, you must create a custom option that contains a compute that toggles the Viewer function *setHelpMode* to on when the form is opened. The following sample shows this compute:

```
<ViewerHelp_on compute="toggle(global.global.activated, &#xA;
   'off', 'on') == '1' ? viewer.setHelpMode('on') : ''">
</ViewerHelp_on>
```

Exceptions To This Practice

There are no exceptions to this practice.

Creating Secure Signatures

This section describes best practices for creating digital signatures and digital signature filters in XFDL forms. These best practices ensure that users are fully aware of the form content they are signing, and assures the non-repudiation of those forms by alerting users of any malicious alteration of the form content. Using best practices for filtering digital signatures creates secure forms that withstand rigorous scrutiny.

These best practices do not attempt to describe every potential scenario related to filtering digital signatures. Instead, it exposes you to sound form-design habits that will strengthen the security of the signatures in your forms.

Topics Discussed

This section discusses the following topics:

- ″Use omit Rather Than keep For Signature Filters″.
- ″Use keep To Create Overlapping Signatures″.
- ″Sign Form and Page Global Options″.
- ″Use Custom Items To Store Custom Information″.
- ″Sign Item Positioning Information″.
- ″Use Absolute Positioning″.
- ″Use Unique Scope Identifiers For Items″.
- ″Sign Related Items″.

Note: Many of these security practices discuss related issues. Therefore, we recommend that you read the entire section before creating forms that require high security.

Use omit Rather Than keep For Signature Filters

When creating signature filters, use the *omit* flag rather than the *keep* flag. Digital signatures with *omit* flags sign *everything* in the form except those elements that you specifically exclude.

Why Use This Practice

Consistently using this practice prevents you from accidentally excluding items and options that should be signed. If you don’t sign the proper form items and options, the form is vulnerable to malicious alteration. In turn, this may lead to forms that will not withstand scrutiny.

When setting filter options with the Designer, *omit* is the default flag. Accepting this default ensures that users sign everything except the items and options you specifically exclude.

If you rely on *keep* filters, you must specify each item and option the signature is to sign. This increases the risk that you will miss an item or option you intended to include in the signature. Also, using *keep* filters could subsequently allow someone to add to the form's contents without breaking the signature. This is prevented if you use *omit* filters.

Note: Filters which use the keep or omit flag include signitems, signoptions, signgroups, signdatagroups, signitemrefs, signoptionrefs, and signpagerefs. For more information on filter options and creating digital signatures, see the Creating Signature Buttons in XFDL document.

Example

There are certain options that form designers must always omit in a form. If you create your forms using the Designer, it automatically creates the *signoptions* filter for you. These options are *triggeritem* and *coordinates*. Both of these options are normally updated after the form is signed. For example, *triggeritem* is set when you submit the form, even if you have already signed it.

The following code depicts an *omit* filter that excludes *triggeritem* and *coordinates*:

```
<signoptions>
   <filter>omit</filter>
   <optiontype>triggeritem</optiontype>
   <optiontype>coordinates</optiontype>
</signoptions>
```

This construction creates a filter that excludes a minimum of information from the signed form. In general, this is a good practice when creating filters.

Note: When omitting items from the signature, ensure that you secure their positioning information. For more information, see ″Sign Item Positioning Information″.

Exceptions To This Practice

An exception to this practice may occur when you are filtering multiple and overlapping signatures. You should generally use *omit* for the form's primary signature, but you can use *keep* for a secondary signature if it signs only the primary signature. For more details, see ″Use keep To Create Overlapping Signatures″ and ″Sign Item Positioning Information″

Use keep To Create Overlapping Signatures

When creating multiple and overlapping signatures, you may be able to use *keep* instead of *omit*. If you want a secondary signature to sign the primary signature, you can use *keep* to sign only the primary signature. For example, you could use this strategy if you wanted a manager to sign an employee's information and signature.

Note: We recommend that you also read ″Use omit Rather Than keep For Signature Filters″.

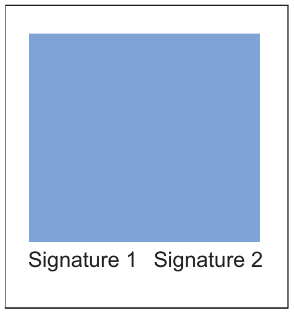
Why Use This Practice

This practice simplifies the creation of overlapping signatures, saving form development time. Your choice of *omit* or *keep* for secondary signatures depends on whether the secondary signatures are signing information the primary signature didn't. In general, when creating secondary signatures you should consider the following:

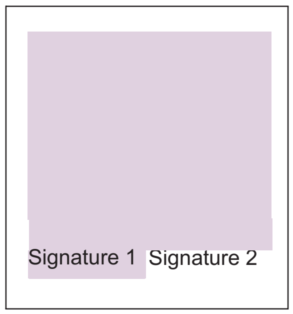
- When the secondary signature signs different information than the primary signature, use *omit* to create the secondary signature filters
- When the secondary signature signs the same information as the primary signature as well as the primary signature itself, you can use *keep* to have the secondary signature sign only the primary signature. Since the primary signature has already locked the contents of the form, the secondary signature only needs to lock the primary signature itself.

Example

The following diagrams show a form in which the first signature signs the body of the form, but not the second signature. The second signature then signs the first signature, and by extensions body of the form. This allows the second signer to endorse the original signature.



Signature 1 signs the filled section of the form.



Signature 2 signs Signature 1 and the filled section of the form.

In the following code sample, SIGNATURE1 and SIGNATURE2 are set as follows:

- `Signature1` is the primary signature.
- `Signature2` is the secondary signature and signs only BUTTON1 and its associated signature.

`<signitemrefs>`
`      <filter>omit</filter>`
`      <itemref>PAGE1.SIGNATURE2</itemref>`
`</signitemrefs>`
	`Signature1` omits `signature2`

`<signoptions>`
`      <filter>omit</filter>`
`      <optiontype>triggeritem</optiontype>`
`      <optiontype>security_info</optiontype>`
`</signoptions>`
	`Signature1` also omits triggeritems, allowing the user to click buttons without breaking the signature.

`<signitemrefs>`
`      <filter>keep</filter>`
`      <itemref>PAGE1.SIGNATURE1</itemref>`
`      <itemref>PAGE1.BUTTON1<itemref>`
`</signitemrefs>`
	`Signature2` keeps `Button1` and `Signature1`

Exceptions To This Practice

There are no exceptions to this practice.

Sign Form and Page Global Options

Set signatures to sign global and page options such as global fonts and background colors. While *omit* filters automatically sign form and page globals unless they are specifically excluded, it is easy to neglect global options when using a *keep* filter. However, modification of non-secure global options can effect the entire form.

Note: Form developers will sometimes omit globals because they contain custom items that need to keep operating after the user has signed the form. However, custom information should be placed in custom items. See ″Use Custom Items To Store Custom Information″ Use Custom Items To Store Custom Information.

Why Use This Practice

If you do not secure your form's global and page options, someone could alter these options to obscure what is being signed. For example, someone could change the background color of a form so that text becomes unreadable.

Global and page options such as global fonts and background colors are not automatically signed by the *keep* filter. As a result, it's best to use *omit*, which automatically secures your global and page options unless you intentionally exclude them from a signature. However, if you must use a *keep* filter, ensure that form and page globals are identified in a *signoptions* or *signoptionrefs* option. (For more details on *omit* and *keep*, see ″Use omit Rather Than keep For Signature Filters″, ″Use keep To Create Overlapping Signatures″, and ″Sign Item Positioning Information″.)

Example

Assume that you have created a one-page form with a grey background color and black font color. At the bottom of the form, you've written a notice in a yellow font that reads, ″By signing this form, you agree to allow XYZ Inc. to sell information about you.″

In this scenario, if you haven't set the signature filters to sign global background and font colors, it's possible for someone to change the default background color from gray to yellow. The new default background would render your notice in yellow font invisible to the reader. This leaves the signed form vulnerable to repudiation and XYZ Inc. exposed to possible legal consequences for selling personal data without permission.

The following code sample shows a *keep* filter that specifically secures the background and font color set in a form global and font information set in a page global. Additionally, it secures custom items placed in the page and form globals

```
<signoptionrefs>
   <filter>keep</filter>
   <optionref>global.global.bgcolor</optionref>
   <optionref>global.global.fontcolor</optionref>
   <optionref>PAGE1.global.fontinfo</optionref>
   <optionref>PAGE1.global.custom_items</optionref>
   <optionref>global.global.custom_items</optionref>
</signoptionrefs>
```

Exceptions To This Practice

There are no exceptions to this practice.

Use Custom Items To Store Custom Information

If you need custom information to update after the user signs the form, place the information in a custom item rather than in the form or page globals. This allows you to sign form and page global options without freezing the custom option.

Why Use This Practice

As discussed in ″Sign Global and Page Options″ Sign Form and Page Global Options, you should set signatures to sign global and page options in a form. However, if you place custom options in a global item, those options are ′locked′ when the global item is signed. This means that the custom options cannot be modified after the user signs the form. As a result, if you need to update your custom information after the user signs the form, you must ensure that the custom information is placed outside of signed global options.

Example

The following sample shows a custom item that contains a custom option and an xfdl option:

```
<custom:event xfdl:sid="STATUS_EVENT"
   xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom">
   <xfdl:active>off</xfdl:active>
   <custom:ID>UF45567/home/users/preferences01</custom:ID>
</custom:event>
```

Exceptions To This Practice

If locking will not affect your custom item’s functionality (that is, if you don’t need to alter the value after it has been signed), you do not need to follow this practice. However, it is still considered good practice to create custom items for storing custom information.

Sign Item Positioning Information

Item positioning specifies where each item is displayed on a form. When creating signature filters, always ensure that you secure the positioning information of all of your form items. Even items that you have *omitted* from your *signitems* option should have a *signoptions* or *signoptionrefs* option filter that keeps their item positions. While *itemlocation* is the most obvious option that contains positioning information, there are other options that could potentially effect the location, size, or appearance of an item, such as size, visible, or scrollhoriz and scrollvert. The Designer automatically secures these options, unless you choose otherwise.

There are two types of positioning information: absolute and relative. Absolute positioning requires that you position items using *x* and *y* coordinates in pixels. The relative positioning scheme, however, places items in relation to other items. If you use relative positioning, you must sign the *anchor item* so that it cannot be moved. If someone moves an un-secured anchor item, it automatically displaces all of the items that are dependent upon it for positioning information. Regardless of the type of item positioning you use, you must secure the positioning information or a malicious user could displace form items and obscure important information.

Using this practice allows you to lock down the appearance and location of items that otherwise cannot be signed. For example, you might exclude certain compute items with an *omit* filter. However, to prevent malicious modification of those items, you can specify options to secure using the *keep* filter. For more information regarding *keep* filters and *itemlocation*, see "Sign Item Positioning Information"

Note: You should set signatures to sign all item positioning information, regardless of whether positioning is absolute or relative. However, whenever practical, you should use absolute rather than relative positioning. For more information, see "Use Absolute Positioning".

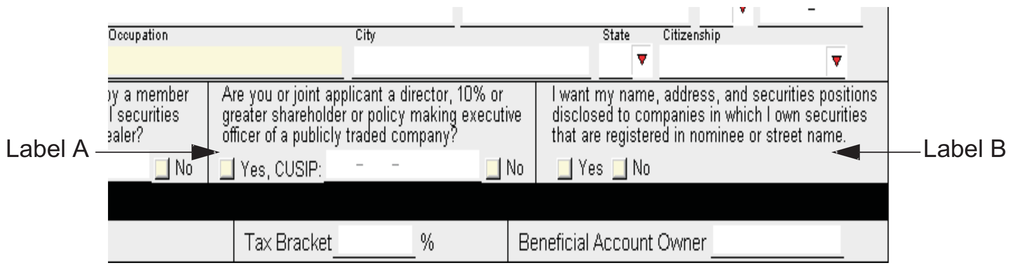
Why Use This Practice

Setting signatures to sign positioning information helps prevent someone from using unsigned items and options to obscure signed items. If items are obscured, readers could potentially misunderstand what they are signing. This may allow users to repudiate their signed forms based on an incorrect perception of what they were signing.

Consider an online insurance policy application. If a flaw in a signature filter permitted someone to obscure contract terms that are crucial to the user's understanding, the contract could be successfully challenged.

Example

The following diagrams illustrate a portion of a brokerage application form. In this form, the check boxes and fields are positioned below the anchor labels using relative positioning. These anchor labels are not secured by a signature filter. As you can see in the second diagram, LabelB and its dependent items have been moved to obscure LabelA and its dependent items.



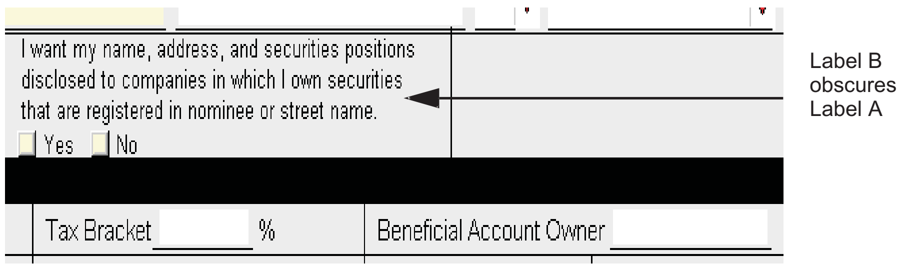
Occupation City State Citizenship

...y a member ...l securities ...ealer? No

Are you or joint applicant a director, 10% or greater shareholder or policy making executive officer of a publicly traded company?
Yes, CUSIP: No

I want my name, address, and securities positions disclosed to companies in which I own securities that are registered in nominee or street name.
Yes No

Tax Bracket %
Beneficial Account Owner



I want my name, address, and securities positions disclosed to companies in which I own securities that are registered in nominee or street name.
Yes No

Tax Bracket %
Beneficial Account Owner

For example, in the following code the signature filter omits LABELB but keeps its itemlocation:

`<signitemrefs>` `<filter>omit</filter` `<itemref>PAGE1.LABELB</itemref>` `</signitemrefs>`	This signitemrefs option omits LABELB from the signature filter.
`<signoptionrefs>` `<filter>keep</filter>` `<optionref>PAGE1.LABELB.itemlocation</optionref>` `<optionref>PAGE1.LABELB.size</optionref>` `<optionref>PAGE1.LABELB.fontinfo</optionref>` `<optionref>PAGE1.LABELB.visible</optionref>` `<optionref>PAGE1.LABELB.value</optionref>` `</signoptionrefs>`	This signoptionrefs option secures LABELB's itemlocation.

Usage Notes

If you use an *omit* signature filter, you automatically secure the form's item positioning, except for those items you specifically exclude. If you do exclude some items with an *omit* filter, pair it with a *keep* filter to secure the item positioning information.

Itemlocation is not the only option that can potentially affect the location, size, or appearance of an item. If an item is omitted from the signature filter, you should consider adding all of these options to a keep filter. These options include:

- itemlocation
- fontinfo
- labelfontinfo
- size
- visible
- value
- scrollhoriz
- scrollvert
- fontcolor
- labelfontcolor
- bordercolor
- labelborderwidth
- thickness

Exceptions To This Practice

There are no exceptions to this practice.

Use Absolute Positioning

Absolute positioning allows you to precisely place items by setting their *x* and *y* coordinates in pixels. Each absolutely positioned item contains its own location and extent information and is therefore independent of other items on the form. On the other hand, items with relative positioning are positioned in *relation* to other items on the form. Their *itemlocation* information is dependent upon an *anchor item*. If you move the anchor item, all of its dependent items move to maintain their positions relative to the anchor item.

Why Use This Practice

Relative positioning affects more than the location of items. Relative positioning can also affect the alignment and size of items. Any adjustments to the location or size of an *anchor item* could potentially affect numerous other items. Additionally, to secure the *itemlocation* of a relatively positioned item, you must sign the positioning of the *anchor item*.

Using absolute positioning guarantees that the location and extent of items are not reliant on other items. Each item contains its own *itemlocation* information, making signature filtering easier and more precise. Therefore, forms with absolute positioning are more difficult to compromise.

An item's own *itemlocation* code is the only factor in determining its absolute positioning. On the other hand, the same item positioned relatively could depend on several other *anchor items*, which in turn could be dependent on still other items. Displacement of a single *anchor item* could result in a domino effect of misplaced and mis-sized items.

Example

The following code samples compare the *itemlocation* information of an identical button. The first sample uses relative positioning, while the second uses absolute positioning.

```
<button sid="BUTTON1">
   <value>Print</value>
   <itemlocation>
      <after>LABEL1</after>
      <expandl2c>BUTTON3</expandl2c>
      <offsetx>-94</offsetx>
      <offsety>70</offsety>
   </itemlocation>
</button>
```

Relative positioning depends upon a number of variables. Items may even be anchored to more than one anchor item. For example, `label1` and `button3`

```
 <button sid="BUTTON1">
   <value>Print</value>
   <itemlocation>
       <x>200</x>
      <y>200</y>
      <width>130</width>
      <height>22</height>
   </itemlocation>
</button>
```

Absolutely positioned items contain all of their own itemlocation information.

Exceptions To This Practice

If you are transferring form information from a database table with an indefinite number of rows, you must use relative positioning. Moreover, some forms have dynamically created sections, such as a purchase order form that allows users to add multiple rows. In such cases, you must use relative positioning.

Use Unique Scope Identifiers For Items

Scope identifiers, or *sids,* uniquely identify each item within the XFDL code. Each item on a page should have a different sid to differentiate it from other items on that page. You may use the following characters: a-z, A-Z, 0-9, dollar sign ($), and underscore (_). Spaces are not permitted.

Why Use This Practice

Each item should have a unique sid to so that the Viewer and Designer can tell them apart. Older versions of the Viewer allow forms to contain duplicate sids, but treat them unpredictably. For example, computes involving two items that share the same sid, called *duplicate sids*, may behave incorrectly, or may not function at all. As a result, when hand-coding forms you should create unique sids for each item on every page. It is good practice to choose sids that describe the purpose of the item, such as ″Print_Button″.

If you create your forms using the Designer, it automatically creates unique sids by numbering each item as it is created. For example, LABEL1, LABEL2, and so on.

Note: If you open a hand-coded form in Designer 2.0 or later or Viewer 4.5 or later, they automatically flag duplicate sids as errors and will not allow you to use them.

Example

You must establish sids in the first line of an item's code, as shown in the following example:

```
<button sid="Print_Button">
```

However, items on separate *pages* may have duplicate sids. For example, if you had a print button on multiple pages, each button's sid could be ″Print_Button″, as shown below:

```
<page sid="PAGE1">
   <global sid="global"></global>
   <label>PAGE1</label>
   <button sid="Print_Button">
      ...
 </page>

<page sid="PAGE2">
   <global sid="global"></global>
   <label>PAGE2</label>
   <button sid="Print_Button">
      ...
 </page>
```

Duplicate sids on separate pages are permitted because each item's *reference* includes the sid of it's page. Therefore, if a compute involved the first button in the code above, the full reference would be:

```
PAGE1.Print_Button
```

This allows the Viewer to differentiate between buttons with the same name on different pages.

Exceptions To This Practice

There are no exceptions to this practice.

Sign Related Items

Many items point to another item for their values, images, and help messages. These items are considered *related*. Related items include lists, popups, and comboboxes with cells, buttons and images, or any item linked to a help message. For the best security, when you sign an item, you must sign all related items.

If there is a conflict between signed and unsigned items, you must err on the side of signing. For example, you can omit a cell group for a popup, but if two popups use the same cell group and only one popup is signed, then the cell group should still be signed.

Why Use This Practice

If a signature signs only one item in a related pair of items, the unsigned item can be modified. For example, someone could alter unsigned *cell* information even when the related *popup* item has been signed. As a result, the content of the form could be changed after the user has signed it.

While an omit filter automatically signs all items and options that you do not specifically exclude, keep filters must explicitly list all items and options you want to secure. Because related items often do not appear as separate items on the form, it is easy to neglect them in your keep filter. If you are using keep filters, always double-check that you are securing all related items. For example, images are stored in separate data items. Items that display images simply contain references to the data item. Therefore, if you secure a button using a keep filter but forget to secure the image data, the button's image is not secured and could be modified.

Example

Assume you are creating a popup with the following two choices visible to the user:
- ″Please do not sell my personal information″
- ″You may sell my personal information″

When the users sign the form, the popup is secured with a *keep* filter that does not specify the popup's cells.

Once a choice is selected, the scope identifier (SID) of the cell containing the choice is stored in the popup's *value* option. In this example, assume the user chooses ″Please do not sell my personal information″. The SID of this choice is *cell1*. The secured popup stores a reference to cell1 in it's *value* option. This secures the SID of the popup, preventing users from selecting a new cell from the popup. However, the cell itself is not signed. A malicious user could open the form in a text editor and change the text contained in cell1 so that it no longer reflects the choice made by the user. For example, the *cell's* value could be changed to ″You may sell my personal information″. Anyone viewing the form after this alteration would think that the user permitted the company to sell her personal information.

Exceptions To This Practice

There are no exceptions to this practice that you should consider

Note: While there are no exceptions to this practice, you should be aware that conflict resolution between signed and unsigned items must err on the side of signing. For example, you can omit a cell group for a popup, but if two popups use the same cell group and only one popup is signed, then the cell group should still be signed.

Creating Certificate Filters

This section describes practices for creating certificate filters in XFDL forms. These best practices help form designers ensure that users select the appropriate certificate for signing forms.

There are two reasons for filtering certificate information within a form's code:
- To control the certificate information displayed when users select a signature identity
- To limit the number of certificates displayed when users select a signature identity

Typically users have more than one digital certificate on their computers. They acquire the certificates from various Certificate Authorities, using different signature engines, and each certificate may have a different purpose. As a result, the Viewer often presents a long list of signing identities when users want to sign a form. If a form has specific signing requirements, form designers should ensure that the Viewer displays all the information necessary to enable users to select the correct signing identity, while limiting extraneous data. Alternatively, designers could limit certificate choices so that the Viewer only displays applicable signing identities.

There are two types of certificate filters:
- filteridentity
- dialogcolumns

The *dialogcolumns* filter is easiest method of controlling the certificate information that is displayed to the user. It allows you to specify the information displayed to users so that they can easily choose the correct signing certificate. On the other hand, *filteridentity* filters are much more difficult to create for the public, because they filter for precise values. When you create forms for use by the general public you are unlikely to know the details of their digital certificates. *Filteridentity* filters require exact matches, including spelling, case, and punctuation. Unfortunately, CryptoAPI and NSS certificates use different strings to report the same information, preventing an exact match by both signature engines. As a result, certificate filtering for precise values is only useful when the signer's digital certificate contains predictable information. In other words, organizations may choose to specify which certificates sign in-house documents, but it is not practical to filter for specific values when creating forms that will be signed by the public.

This section includes the following topics:
- ″Use RSA-Compatible Engines″.
- ″Use dialogcolumns To Control The Certificate Information Displayed To Users″.
- ″Use filteridentity To Limit The Number Of Signing Identities Displayed To Users″.

For more information on certificate attributes and the values returned by signature engines, see ″Appendix B: Certificate Attributes″.

Use RSA-Compatible Engines

You must use RSA-compatible signature engines if you want to create certificate filters. The only fully RSA-compatible signature engines supported by XFDL are Microsoft CryptoAPI, Netscape, and generic RSA engines. While some aspects of certificate filtering are applicable to the Clickwrap engine, this is only true for the functionality that overlaps with RSA-compatible engines.

Why Use This Practice

If you want to limit the type of certificates that the Viewer displays to the user, you must designate an RSA-compatible engine, such as CryptoAPI, Netscape, or generic RSA. These signature engines allow you to fully implement certificate filtering, by accepting the *filteridentity* and *dialogcolumns* parameters. These parameters are *certificate attributes* common to X.509 digital certificates. Other digital certificate providers do not use X.509 certificates, and as a result, PenOp or Entrust signing engines do not have certificate filtering functionality.

Example

You can specify the type of signature engine that the form should use by adding an *engine* parameter to the *signformat* option. If the *engine* parameter is blank, or not included in the option, the Viewer will automatically use the generic RSA engine, which uses both the Netscape and CryptoAPI engines. For example, the following code samples demonstrate engine parameters that specify RSA-compatible engines.

```
<signformat>application/x-xfdl; engine="CryptoAPI"</signformat>
<signformat>application/x-xfdl; engine="Netscape"</signformat>
<signformat>application/x-xfdl; </signformat>
```

Exceptions To This Practice

There are no exceptions to this practice.

Use dialogcolumns To Control The Certificate Information Displayed To Users

The *dialogcolumns* filter allows you to control the certificate information displayed to users when they are selecting a signature identity.

When creating *dialogcolumns* filters, ensure that the information you want to display to the user actually exists as a certificate attribute. If it does not, the missing attributes are displayed as blank spaces followed by a comma. For more information, see ″Appendix B: Certificate Attributes″.

Why Use This Practice

Most digital certificates contain a lot of information regarding the user, the certificate issuer, and the certificate itself. By default, the signature engines automatically filter this information so that only user names and the expiry dates of the certificate are displayed to users. However, you may find that this is not enough information to assist users in selecting the correct certificate. On the other hand, if you do not filter certificate information you may display data that is irrelevant to users trying to select a certificate. Controlling the information that is displayed makes it easier for users to select the correct certificate they need to sign the form. If you use *dialogcolumns* to filter information, you can choose precisely what information you want to display, assisting users to select the correct certificate quickly and easily.

Example

The following sample shows a *dialogcolumns* filter that limits the certificate information displayed to users to the user name, e-mail address, and department:

`<signdetails>`
`   <dialogcolumns>`
`      <property>Subject: CN</property>`
	Subject: CN (Common Name) — The signer's name.

`      <property>Subject: E</property>`	**Subject: E (Email)** — The signer's e-mail address.

`      <property>Subject: OU</property>`
`   </dialogcolumns>`
`</signdetails>`
	Subject: OU (Organizational Unit) — The signer's department.

In this case, the Viewer's signing identity list only displays the signer's common name, e-mail address, and department of the certificate. If any of these attributes do not exist in a certificate, that property is displayed as a blank.

Exceptions To This Practice

There are no exceptions to this practice.

Use filteridentity To Limit The Number Of Signing Identities Displayed To Users

The *filteridentity* filter allows you to limit the number of certificates displayed to users when they are selecting a certificate.

Why Use This Practice

Users frequently have multiple digital certificates on their computers. While these certificates may contain identical user information, their issuer and certificate information may be completely different. By default, the signing ceremony displays all of a user's signing identities, but if those certificates have different purposes, it may be difficult for users to choose the appropriate one for signing a specific form.

Filtering the certificates limits the number of signing identities listed by the Viewer. This allows you to specify attributes or attribute values that the certificate must have. If a certificate does not have the listed attributes or values, it is not included in the list of signing identities. This creates a more manageable list, making it easier for users to select the correct certificate for signing the form.

Example

The following code sample depicts a *filteridentity* filter that specifies the user's department (organizational unit) and designates it as IT.

```
<filteridentity>
   <filter>
   <tag>Subject: OU</tag>
   <value>IT</value>
   </filter>
</filteridentity>
```

In this case, the Viewer’s signing identity list only displays certificates that have an organizational unit parameter with a value of IT. If a certificate does not contain an attribute with this value, it is not included in this list.

Note: The value parameter may contain a value or remain blank. If you insert a value, the attribute value in the certificate must match. If it contains a value, the signature engine attempts to match that value with the value of the Subject: OU certificate attribute. If it is blank, the signature engine verifies that the attribute exists.

Usage Notes

When creating *filteridentity* filters, you must ensure that the required attributes match an existing certificate. If you are overly explicit with your certificate filtering criteria, you may add a certificate attribute or value that does not exist in any user certificate. If no certificates match the specified certificate attributes, then the Viewer will display an empty signing identity list and the user will be unable to sign the form. For example, if your *filteridentity* filter specifies that users must use CryptoAPI certificates, then users with only Netscape certificates will be unable to sign your form.

Consider the following tips when creating a *filteridentity* filter:
- All of the listed attributes and values must exist in the user’s digital certificate.
- You cannot use ’if’, ’or’ or ’not’ statements unless you use a compute.
- Ensure that the spelling, case, and punctuation of you attribute list exactly match the user’s digital certificate.

Exceptions To This Practice

There are no exceptions to this practice.

Appendix A: Screen Reader Announcements

This appendix presents the default text read by each of the three supported screen readers:

- JAWS
- Microsoft Narrator
- Microsoft Eyes

This information will help you create more meaningful accessibility messages for users who rely on screen readers.

Placeholder Conventions

The following placeholders are used to represent custom information that is part of the form's design:

<*acclabel*>
: The item's accessibility message, defined by the associated *acclabel* item.

<*choice*>
: One of the selections in a list, defined by a *cell* item.

<*contents*>
: The text that appears inside the field, contained in the item's *value* option.

<*button value*
: The text that appears on the button, defined by the item's *value* option.

<label>
: The text that appears in the item's label, defined by the item's *value* option.

<invalid alert>
: The alert message that appears if a user tabs out of a field that contains an invalid entry.

x Indicates the position a choice has in a list. For example, **1** of 7.

n Indicates the number of choices in a list. For example, 1 of **7**.

<help>
: The help message that only appears if Viewer Help is on and:
  - The item contains the focus indicator.

  or

  - The item has been passed over by the mouse.

The following tables shows the text that the screen readers announce for each type of interactive form item in response to different user actions.

XFDL Item	User Action	JAWS	Narrator	Eyes
Field: Empty read/write	Move to item using tab key. Move to item using shift+tab.	*<label>* *<acclabel>* Edit field is empty. *<help>* Edit. Type in text.	*<label>* *<acclabel>* Field is empty. *<help>* Editable text.	Editbox *<label>* *<acclabel>* Edit field is empty. *<help>*

XFDL Item	User Action	JAWS	Narrator	Eyes
Field: Read/write with contents	Move to item using tab key. Move to item using shift+tab.	*<label> <acclabel>* Edit field contains *<contents>*. *<help>* Edit. Type in text.	*<label> <acclabel>* Field contains *<contents>*. *<help>* Editable text.	Editbox *<label> <acclabel>* Edit field contains *<contents>*. *<help>*
Field: Read-only containing text	Move to item using tab key. Move to item using shift+tab.	*<label> <acclabel>* Edit field contains *<contents>*. Read-only. *<help>* Edit. Type in text.	*<label> <acclabel>* Field contains *<contents>* Read-only. *<help>* Editable text	Editbox *<label> <acclabel>* Edit field contains *<contents>*. Read-only. *<help>*
Field: Write-only containing text	Move to item using tab key. Move to item using shift+tab.	*<label> <acclabel>* Edit field contains *<contents>*. Write-only. *<help>* Edit. Type in text.	*<label> <acclabel>* Write-only. *<help>* Editable text	Editbox *<label> <acclabel>* Write-only. *<help>*
Field: Formatted read/write with invalid entry	Tab out of item that contains an invalid entry.	<invalid alert> <help> Edit.	Window *<invalid alert> <help>* Window	Window *<invalid alert> <help>*
List: Empty read/write	Move to item using tab key. Move to item using shift+tab.	Form pane. Listbox. *<label> <acclabel>* Nothing selected. *<help>* Listbox. *n* of *x*. To move to an item, press the arrow keys.	*<label> <acclabel>* Nothing selected. *<help>* List.	Listbox.*<label> <acclabel>* Nothing selected. *<help>*
List: With selection	Move to item using tab key. Move to item using shift+tab.	Form pane. Listbox. *<label> <acclabel> <choice>* selected. *<help>* Listbox. *n* of *x*. To move to an item, press the arrow keys.	*<label> <acclabel> <choice> <help>* List.	Listbox.*<label> <acclabel> <choice> <help>*
List: Activating list	Press spacebar or use down arrow to scroll through choices in list.	*<label> <acclabel> <choice>* selected. *<help>* Listbox. *<choice>*	*<choice>* list item.	*<choice>* list item.
List: Moving through list	Up or down arrow key.	<choice>	*<choice>* list item.	*<choice>* list item.

XFDL Item	**User Action**	**JAWS**	**Narrator**	**Eyes**
List: Read-only list	Move to item using tab key. Move to item using shift+tab.	Form pane. Listbox. *<label>* *<acclabel>* *<choice>* selected. Read-only. *<help>* Listbox. *n* of *x*. To move to an item, press the arrow keys.	*<label>* *<acclabel>* *<choice>* Read-only. *<help>* List.	Listbox.*<label>* *<acclabel>* *<choice>* Read-only. *<help>*
List: Formatted list	Move to item using tab key. Move to item using shift+tab.	Form pane. Listbox. *<label>* *<acclabel>* *<choice>* selected. *<help>* Listbox. *n* of *x*. To move to an item, press the arrow keys.	*<label>* *<acclabel>* *<choice>* *<help>* List.	Listbox.*<label>* *<acclabel>* *<choice>* *<help>*
Popup: Empty	Move to item using tab key. Move to item using shift+tab.	*<acclabel>* *<label>* This is a popup list. Use the space bar or down arrow key to bring up the list. *<help>* Combobox. 0 items. To change the selection, use the arrow keys.	*<acclabel>* *<label>* Nothing selected. This is a popup list. Use the space bar or down arrow key bring up the list. <help> Combobox.	Combobox. *<acclabel>* *<label>* This is a popup list. Use the space bar or down arrow key to bring up the list. *<help>*
Popup: Activating list	Press the spacebar or down arrow key.	List box. *<choice>* *n* of *x* To move to an item, press the arrow keys.	Foreground window list. List. *<choice>* List item.	Static box. Listbox. *<label>* *x* of *n*
Popup: Moving through list	Press the arrow keys.	*<choice>*	*<choice>* List item.	*x* of *n* Note: Windows® Eyes does not identify the list choice as the user proceeds down the list. It merely lists it's position.
Popup: With selection	Move to item using tab key. Move to item using shift+tab.	*<acclabel>* *<choice>* This is a popup list. Use the space bar or down arrow key to bring up the list. *<help>* Combobox. 0 items. To change the selection, use the arrow keys.	*<acclabel>* *<choice>* selected. This is a popup list. Use the space bar or down arrow key to bring up the list. *<help>* Combobox.	Combobox. *<acclabel>* *<choice>* selected. This is a popup list. Use the space bar or down arrow key to bring up the list. *<help>*

XFDL Item	User Action	JAWS	Narrator	Eyes
Popup: Read-only list	Move to item using tab key. Move to item using shift+tab.	*<acclabel>* *<choice>* This is a popup list. Use the space bar or down arrow key to bring up the list. Read-only. *<help>* Combobox. 0 items.	*<acclabel>* *<choice>* This is a popup list. Use the space bar or down arrow key to bring up the list. Read-only. *<help>* Combobox.	Combobox. *<acclabel>* *<choice>* selected. This is a popup list. Use the space bar or down arrow key to bring up the list. Read-only. *<help>*
Combobox: Empty	Move to item using tab key. Move to item using shift+tab.	*<label>* *<acclabel>* is empty. This is an editable combobox. Type the text or use the down arrow key to choose from the list. *<help>* Combobox. 0 items. To change the selection, use the arrow keys.	*<label>* *<acclabel>* contains. This is an editable combobox. Type the text or use the down arrow key to choose from the list. *<help>* Combobox.	Combobox. *<label>* *<acclabel>* is empty. This is an editable combobox. Type the text or use the down arrow key to choose from the list. *<help>* .
Combobox: After activating list	Press the spacebar or down arrow key.	Listbox. Not selected. *<choice>* To move to an item, press the arrow keys.	Foreground window. List. List. *<choice>* List item.	Static box. Listbox. *<choice>* *x* of *n*
Combobox: Moving through list	Press the arrow keys.	*<choice>*	*<choice>* List item.	*<choice>* *x* of *n*
Combobox: With selection	Move to item using tab key. Move to item using shift+tab.	*<label>* *<acclabel>* contains *<choice>* This is an editable combobox. Type the text or use the down arrow key to choose from the list. *<help>* Combobox. 0 items. To change the selection, use the arrow keys.	*<label>* *<acclabel>* contains *<choice>* This is an editable combobox. Type the text or use the down arrow key to choose from the list. *<help>* Combobox.	Combobox. *<label>* *<acclabel>* contains *<choice>*. This is an editable combobox. Type the text or use the down arrow key to choose from the list. *<help>*

XFDL Item	**User Action**	**JAWS**	**Narrator**	**Eyes**
Combobox: Read-only list	Move to item using tab key. Move to item using shift+tab.	*<label>* *<acclabel>* contains *<choice>*Use the spacebar or down arrow key to bring up the list. To activate, press spacebar. Read-only. *<help>*	*<label>* *<acclabel>* contains *<choice>* This is an editable combobox. Type the text or use the down arrow key to choose from the list. Read-only.*<help>* Combobox.	Combobox. *<label>* *<acclabel>* contains *<choice>*. This is an editable combobox. Type the text or use the down arrow key to choose from the list. Read-only. *<help>*
Combobox: Write-only list	Move to item using tab key. Move to item using shift+tab.	*<label>* *<acclabel>* contains *<choice>*Use the spacebar or down arrow key to bring up the list. To activate, press spacebar. Write-only. *<help>*	*<label>* *<acclabel>* contains *<choice>* This is an editable combobox. Type the text or use the down arrow key to choose from the list. Write-only. *<help>* Combobox.	Combobox. *<label>* *<acclabel>* contains *<choice>*. This is an editable combobox. Type the text or use the down arrow key to choose from the list. Write-only. *<help>*
Combobox: Formatted list with invalid entry	Tabbing out of list with an invalid entry.	Combobox. *<invalid alert>* *<help>* Combobox.	Window *<invalid alert>* *<help>* Window	Window *<invalid alert>* *<help>*
Check: Not selected	Move to item using tab key. Move to item using shift+tab	*<label>* *<acclabel>* Checkbox not checked. *<help>* Checkbox not checked. To check, press spacebar.	*<label>* *<acclabel>* Checkbox not checked.*<help>* Checkbox. To check, press spacebar.	Checkbox. Unchecked. *<label>* *<acclabel>* Checkbox not checked. *<help>*
Check: Selected	Move to item using tab key. Move to item using shift+tab	*<label>* *<acclabel>* Checkbox. checked. *<help>* Checkbox. checked. To clear checkmark, press spacebar.	*<label>* *<acclabel>* Checkbox checked.*<help>* Checkbox. To check, press spacebar.	Checkbox. Checked. *<label>* *<acclabel>* Checkbox checked. *<help>*
Radio: Not selected	Move to item using tab key. Move to item using shift+tab	*<label>* *<acclabel>* Radio button. *x of n*. Not selected. *<help>* Radio button not checked. To change the selection, press up or down arrow.	*<label>* *<acclabel>* Radio button. <*x* of *n*> Not selected. *<help>* Radio button. To select a different item in the cluster, use the arrow keys.	Radio button. Unchecked. *<label>* *<acclabel>* Radio button. <*x* of *n*> Not selected. *<help>*

XFDL Item	User Action	JAWS	Narrator	Eyes
Radio: Selected	Move to item using tab key. Move to item using shift+tab.	*<label>* *<acclabel>* Radio button. *x of n* selected. *<help>* Radio button checked. To change the selection, press up or down arrow.	*<label>* *<acclabel>* Radio button. <*x* of *n*> selected. *<help>* Radio button. To select a different item in the cluster, use the arrow keys.	Radio button. Checked. *<label>* *<acclabel>* Radio button. <*x* of *n*> Selected. *<help>*
Button	Move to item using tab key. Move to item using shift+tab.	Button. *<button value>* *<acclabel>* *<help>* Button. To activate, press spacebar.	Button. *<button value>* *<acclabel>* *<help>* To press, use spacebar.	Button. *<button value>* Button. *<acclabel>* *<help>*

Appendix B: Certificate Attributes

This appendix lists the filterable certificate attributes and indicates whether the CryptoAPI and Netscape signature engines return identical strings when returning certificate attributes with identical data. If you want to filter for specific *filteridentity* tag values without specifying an engine, filter for strings that can be recognized by both RSA-compliant signature engines. If you want to filter for other tag values, your organization should maintain a policy stating the preferred signature engine for attaining digital certificates.

For example, if you wanted to filter for certificates encrypted with an md5 algorithm, you would find that Netscape and CryptoAPI return different strings for this attribute. CryptoAPI returns *md5RSA*, while Netscape returns *PKCS #1 MD5 With RSA Encryption*. If you listed either of these attributes as a tag value, you would return certificates from only one engine.

Note: This is not an issue for strings that are expected to be unique from certificate to certificate, such as the certificate serial number or public key.

Certificate Attribute	Description	Identical Strings?
Subject: CN	the certificate owner's common name	Yes
Subject: E	the certificate owner's e-mail address	Yes
Subject: T	the certificate owner's locality	Yes
Subject: ST	the certificate owner's state of residence	Yes
Subject: O	the organization to which the certificate owner belongs	Yes
Subject: OU	the name of the organizational unit to which the certificate owner belongs	Yes
Subject: C	the certificate owner's country of residence	Yes
Subject: STREET	the certificate owner's street address	Yes
Subject: ALL	the certificate owner's complete distinguished name	Yes
Issuer: CN	the certificate issuer's common name	Yes
Issuer: E	the certificate issuer's e-mail address	Yes
Issuer: T	the certificate issuer's locality	Yes
Issuer: ST	the certificate issuer's state of residence	Yes
Issuer: O	the organization to which the certificate issuer belongs	Yes

Certificate Attribute	Description	Identical Strings?
Issuer: OU	the name of the organizational unit to which the certificate issuer belongs	Yes
Issuer: C	the certificate issuer's country of residence	Yes
Issuer: STREET	the certificate issuer's street address	Yes
Issuer: ALL	the certificate issuer's complete distinguished name	Yes
Serial	the certificate's serial number	No
SignatureAlg	the algorithm used by the Certificate Authority to sign the certificate	No
BeginDate	the date at which the certificate becomes valid	Yes
EndDate	the date at which the certificate becomes invalid	Yes
PublicKey	the certificate's public key	No
FriendlyName	the certificate's friendly name	No
KeyUsage: ALL	indicates the purposes for which the certificate's public key can be used	No
KeyUsage: Digital Signature	this certificate's public key can create digital signatures	No
KeyUsage: NonRepudiation	this certificate's public key can be used for non-repudiation	No
KeyUsage: KeyEncipherment	this certificate's public key can encipher keys	No
KeyUsage: DataEncipherment	this certificate's public key can encipher data	No
KeyUsage: KeyAgreement	this certificate's public key can ensure that other public keys match their certificates. Used in certificate management.	No
KeyUsage: KeyCertSign	this certificate's public key can sign key certificates	No
KeyUsage: CRLSign	this certificate's public key can sign Certificate Revocation Lists	No
KeyUsage: EncipherOnly	this certificate's public key can only encipher keys or data	No
KeyUsage: DecipherOnly	this certificate's public key can only decipher keys or data	No
BasicConstraints	behaves as though the fCA tag was specified	Yes
BasicConstraints: fCA	determines whether the subject of this certificate can act as a Certificate Authority (1 if true, 0 if false)	Yes

Certificate Attribute	Description	Identical Strings?
BasicConstraints: pathLength	the number of CA certificates that can follow this certificate in a certification path.	Yes
Policies	returns all of the Object Identification Numbers's of the certificate's policies in a comma separated string	Yes
PolicyConstraints: requireExplicitPolicy	indicates whether an explicit policy is required	Yes
PolicyConstraints: inhibitPolicyMapping	indicates whether policy mapping is inhibited	Yes
Engine: Name	the name of the signature engine that created the certificate	Yes

Appendix C: Additional Usage Notes

This appendix lists additional minor form design issues for Workplace Forms Viewer:

- XForms Soap submissions must include the *mediatype*, followed by an optional *charset* attributes, followed by an optional action. You must specify the *mediatype*, *charset*, and *action* attributes in the order indicated. If you reverse the order of *charset* and *action*, then the HTTP charset header will receive the value of the action set in the form, and the action header will receive the value of the form charset.
- The only valid *charset* parameter for XForm Soap submissions is UTF-8.
- Workplace Forms Viewer does NOT support the following:
  - Using the *next* option to direct the focus to a new page.
  - Justify for images in buttons and labels (images are centered).
  - A *printpages* option with a *borderwidth* setting greater than 1. If integers greater than 1 are used, the *borderwidth* around pages will remain at 1 pixel.
- The Viewer expects the *fontinfo* option's array elements to be listed in the following order:
  1. font name
  2. point size
  3. any other desired elements
- If you bind input constraints on a string, predictive type checking will not work.
- Radio button values cannot contain a compute. However, they can be modified by the *set* function, and you can reference radio button values in computes for other options.
- If using a *set* function to set the on/off values for radio buttons, it is possible to destroy the functionality of the radio button's group option. To avoid this, set all radio buttons in a particular group at the same time. For example, if you use a *set* function to turn a radio on, make sure you use a set function to turn each of the other radio buttons off.
- Because of the nature of web browsers, a user can always return to a cancelled form. For example, if you give a button a type of "done", the form will perform a submission and then cancel itself. However, in a web browser, a user can click the Back button to return to the form.
- The *destroy* function cannot delete the item that triggered the destroy. For example, if clicking a button calls the destroy function, the function cannot destroy the button itself.

  Additionally, you cannot use the XFDL *destroy* function to remove a spacer from the form. If you attempt to do this, the Viewer will not destroy the spacer and will display an error.
- If a field's *label* option is empty and you set the *label* option dynamically (with a compute), the field is not redrawn properly.
- When you close a form, the *activated* option for the form and any page in the form (that is global.global.activated and page.global.activated) do not turn off before the form closes.
- Signing items with fonts that are 5 points or smaller in size may cause the signature to break across different versions of Windows.

The Viewer uses the Windows “Small Font” when displaying any fonts that are 5 points or smaller in size. The metrics used to display the Small Font change across different versions of Windows, and may result in slightly larger or smaller text depending on the operating system.

If your items are sized by character size or if they default to the length of the text, this may result in slightly larger or smaller items, which will cause the layout test to fail when validating a signature.

To correct this problem, ensure that all items using small fonts are sized using the extent setting in the *itemlocation* option. This will size the item based on the pixel size rather than character size, ensures the items do not change their size based on the font in use.

- Signing items with fonts that are not generally available may cause the signature to break across different computers.

  If the font is not available on the current computer, Windows will substitute a different font that may not be exactly the same size.

  If your items are sized by character size or if they default to the length of the text, this may result in slightly larger or smaller items, which will cause the layout test to fail when validating a signature.

  To correct this problem, ensure that all items using uncommon fonts are sized using the extent setting in the itemlocation option. This will size the item based on the pixel size rather than character size, ensures the items do not change their size based on the font in use.
- The Viewer does not support dynamically changing RTF fields to plain text. Attempting to do so may result in unexpected behavior.
- When printing rich text fields, background colors will not print properly. To solve this problem, you should set rich text fields to have a *printbgcolor* option of white.

  If you are using Rich Text fields as labels and want them to match the background color of the form, consider printing the entire form with a white background. This will print a clean looking form and will save toner.
- Do not use rich text in fixed size fields. Rich text does not support the level of precision required to properly limit the text within the constraints of the field. In many cases, formatting and printing will cause the text to extend beyond the boundaries of the field.
- Rich text fields do not support a high level of precision when printing. This means that the text may look different or wrap differently when printed.
- Do not use F1 or F6 as keypress events. When the Viewer is running in a web browser, the browser interferes with capturing these events.
- If the endianness of a form is important to your Workplace Forms implementation, ensure you specify the endianness within your forms (encoding="UTF-16LE" or encoding="UTF-16BE"). If you do not specify the endianness of a form, the form’s endianness is based on the system on which the form is being modified, which may result in the endianness of the form changing. For example, if the form is notarized on a Windows system (little-endian) and written to a Solaris system (big-endian), the endianness of the form will change from little-endian to big-endian.
- The use of the x-xfdl MIME type is no longer recommended. However, this version of the Viewer continues to support this MIME type to maintain backward compatibility with older forms. Use the application/vnd.xfdl MIME type instead.

- Workplace Forms Viewer prints the body of the form, including the form itself and all information in the form. However, the Viewer does not print the toolbar of the form, nor can it print only the data entered into the form.
- Workplace Forms Viewer refuses to display some forms containing “fatal” form design errors. These are generally errors that could seriously affect the form’s appearance or functionality.
- Boxes are automatically drawn in the background regardless of their position in the XFDL build order. This means that items framed by a box will be visible and will function properly even if they come after the box in the build order.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Office 4360
One Rogers Street
Cambridge, MA 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX
IBM
Workplace
Workplace Forms

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Program Number:

Printed in USA

S325-2603-00