



**Best Practices Guide**

**Note**

Before using this information and the product it supports, read the information in "Notices," on page 61.

**First Edition (September 2006)**

This edition applies to version 2.6.1 of IBM Workplace Forms Server — Webform Server (product number L-DSED-6RFRFZ and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2003, 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Introduction</b> . . . . .	<b>1</b>
About Webform Server . . . . .	1
Who Should Read This Document . . . . .	1

## **Differences Between Webform Server and Workplace Forms Viewer** . . . . . **3**

Designing Forms for Webform Server . . . . .	3
Action Items . . . . .	3
Appearance of Forms . . . . .	3
Attachments . . . . .	3
Computing URLs. . . . .	4
Dates . . . . .	4
e-mailing Forms . . . . .	4
Event Model . . . . .	4
IFX Files. . . . .	4
Lists . . . . .	4
Locales . . . . .	5
Navigating Forms . . . . .	5
Printing . . . . .	5
Signatures . . . . .	5
Type Checking and Predictive Input Checking . . . . .	6
URLs . . . . .	6
Viewer Settings . . . . .	6
XForms . . . . .	7
XForms Submissions. . . . .	7
XML Data Model . . . . .	7
Other Differences. . . . .	7

## **General Best Practices** . . . . . **9**

Authenticate Users Through Common Practices . . . . .	9
Do Not Rely on the Size of a Field to Limit Input . . . . .	10
Use Help Messages. . . . .	10
Design Forms for Print to Use the Same Page Size . . . . .	11
Compress All Forms . . . . .	12
Portals: Do Not Use Cancel Buttons . . . . .	13
Portals: Disable Unnecessary Toolbar Buttons . . . . .	13
Do Not Use Rich Text . . . . .	14
Test Forms Thoroughly . . . . .	15

## **Creating Dynamic Webform Server Forms** . . . . . **17**

Minimize the Number of Full Page Refreshes . . . . .	17
Restrict Use of Action Items . . . . .	18
Use Page Breaks for Dynamic Form Design. . . . .	19

Do Not Toggle Computes Off Of Event Model Options. . . . .	20
Toggle Computes off of the Triggeritem . . . . .	20
Use Action Items to Trigger Computes Off Of a Page Flip . . . . .	22

## **Choosing the Right Fonts** . . . . . **25**

Use Only Fonts that are Installed on Client Computers . . . . .	25
Do Not Use Symbol or Wingdings Fonts . . . . .	26
Use Matching Windows and Mac Fonts . . . . .	27
Recommended Japanese Fonts . . . . .	27

## **Formatting Fields** . . . . . **29**

Formatting Phone Numbers . . . . .	29
Formatting Postal Codes . . . . .	30
Formatting E-Mail Addresses . . . . .	32

## **Designing Accessible Forms** . . . . . **35**

Why Create Accessible Forms? . . . . .	35
Topics Discussed . . . . .	35
Other Resources . . . . .	35
Provide Appropriate Accessibility Messages . . . . .	36
Put Label Text Into Acclabels . . . . .	42
Use Field Items To Display Text Information . . . . .	43
Assist JAWS Users to Enter Forms Mode . . . . .	45
Place Graphics Inside Buttons . . . . .	45
Minimize and Explain the Use of Dynamic Content . . . . .	47
Reset the Form's Tab Order . . . . .	47
Identify Row and Column Headings . . . . .	48
Use Contrasting Page Background Colors . . . . .	49
Do Not Use List Items. . . . .	50
Avoid Using Write-Only Fields . . . . .	51
Test Forms Thoroughly . . . . .	51

## **Appendix A: Supported XFDL Options** **53**

## **Appendix B: JAWS Announcements** . . . **57**

## **Appendix C: Additional Usage Notes** **59**

<b>Appendix. Notices</b> . . . . .	<b>61</b>
Trademarks . . . . .	62



---

## Introduction

This document describes *best practices* for designing forms for use with Webform Server. These best practices describe both design requirements for forms used with Webform Server and recommendations for creating forms for users with disabilities.

This document divides these 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. The following information is provided with each best practice:

- Overview of the practice.
- Explanation and background information for the practice.
- Examples and usage notes.
- Any exceptions to the practice that may exist.

---

## About Webform Server

Webform Server enables users to view and complete XFDL documents using a standard web browser, without requiring any additional software or plugins. In addition, Webform Server complies with Section 508 regulations by supporting most of the accessibility features included with Microsoft® Windows® 2000 and Internet Explorer 6.0.

For persons with vision disabilities, Webform Server supports JAWS for Windows screen reading utility. Provided that the design of the form follows the practices described in this manual, users with visual impairments will be able to access all of a form's information and functionality.

**Note:** Refer to IBM® *Workplace Forms™ Webform Server Administrator's Guide* for more information about system requirements.

---

## Who Should Read This Document

You should read this document if you need to develop forms that will work with Webform Server. This document is intended for form developers who are familiar with:

- General form design principles
- Extensible Forms Description Language (XFDL)
- Workplace Forms Designer

For more information about XFDL, refer to the *XFDL Specification*. For more information about the Designer refer to *Workplace Forms Designer User's Manual*.



---

## Differences Between Webform Server and Workplace Forms Viewer

Webform Server allows users to complete and submit forms without the need for any client-side software other than a web browser. However, in the absence of specialized client-side software, Webform Server cannot support the full-range of functionality that is offered by Workplace Forms Viewer.

In many cases, these differences in functionality require a different approach to form design. For instance, forms designed for use with the Viewer may include rich text fields and computes that rely on the keypress or mouseover events. However, since neither of these features is supported by Webform Server, these forms would not work in a Webform Server environment.

As a general rule, any form that works with Webform Server will also work with the Viewer, but the reverse is not true. If you are designing forms for an environment that uses both Webform Server and the Viewer, be sure to restrict the functionality of the forms to those features that Webform Server supports.

---

## Designing Forms for Webform Server

If you are a form designer accustomed to designing forms for the Viewer, it is important that you familiarize yourself with the differences between Webform Server and the Viewer. Some of the differences are simple and have obvious implications, while others are more complex and may require you to adopt different strategies when designing forms.

Read the sections below for a quick overview of the primary differences between Webform Server forms and Viewer forms.

### Action Items

Webform Server supports actions, but with some limitations. Webform Server instructs the browser to automatically refresh a form whenever the user interacts with the form in certain ways. As a side effect of a refresh, actions items that are set to occur once will actually occur each time the form is refreshed. Furthermore, actions that are set to repeat will not work properly. Because of this, we recommend extremely limited use of action items. See "Restrict Use of Action Items" .

### Appearance of Forms

Webform Server draws all forms using standard HTML widgets when drawing check boxes, radio buttons, popups, and comboboxes. This may produce a slightly different look when compared to forms displayed in the Viewer. However, this does not change the functionality of the form. We recommend that you test your form in the Viewer and with Webform Server (on all platforms that the form will be viewed on).

### Attachments

Webform Server allows users to attach files to a form in the normal manner. However, users can only attach one file at a time. The attachment dialog will not

allow users to multi-select files. This may dictate slightly different form design, since requiring a large number of attachments can be cumbersome for the user.

## Computing URLs

Webform Server assumes that forms are always submitted to the server that is running Webform Server. If forms are submitted to other servers, Webform Server will not be able to intercept the form and translate it back into XFDL. For this reason, you must make sure that all computed URLs submit the form to the Webform Server.

If you are using Webform Server with Portal, do not use computed URLs.

## Dates

While the Viewer allows you to configure your calendar date preferences, Webform Server does not. Webform Server always interprets ambiguous dates as year, month, day. For example, 01/02/03 would be February 3rd, 2001.

## e-mailing Forms

Webform Server does not allow users to e-mail forms. This means that there is no toolbar control for mailing forms, and that buttons configured with an e-mail URL will not work.

If you need your users to e-mail forms to each other, they must save the forms locally and send them as attachments to regular e-mails. See *IBM Workplace Forms Server — Webform Server Administrator's Guide* for more details.

## Event Model

Computes based on the *activated*, *dirtyflag*, *keypress*, and *mouseover* options do not work under Webform Server.

Computes based on the *focused* and *focuseditem* options will work under Webform Server if the correct settings are modified in the *translator.properties* file. For more information, see "Do Not Toggle Computes Off Of Event Model Options" on page 20.

For detailed information about the *translator.properties* file, see the *IBM Workplace Forms Server - Webform Server Administrator's Guide*.

## IFX Files

Webform Server supports IFX files on the server. That is, your form can reference an IFX file that resides on the server.

Webform Server does not support IFX files on the user's computer (client) or IFX files embedded in a form.

## Lists

If a form contains a list (the XFDL **list** item), and the text within the list is wider than the list box, a horizontal scroll bar will be displayed within the list box when the form is viewed in the Viewer.

If the form is viewed via Webform Server, the list box will not contain a horizontal scroll bar. Any text that does not fit within the list box will be truncated.

## Locales

Webform Server cannot remotely set the user's locale. The locale viewed by the user will be determined by the user's browser, not the form. Furthermore, if you are using the Viewer with a Webform Server application (for example, using Webform Server to deliver XFDL forms embedded within HTML pages for users that have the Viewer installed on their system), the locale of the Viewer may not be the same as the locale of the form. Webform Server cannot specify the locale of the Viewer. The Viewer's locale is always determined based on the Viewer Preferences. The default locale is the locale of the operating system.

## Navigating Forms

The up and down arrow keys will not open popups. Instead, use ALT + UP ARROW and ALT + DOWN ARROW. Once a popup is open you can still use the normal arrow keys to move among the selections. You should also not that pressing the ESC key to leave a popup or combobox does not cancel the last choice the user made before leaving the widget. Although this is the default HTML form response to the ESC key, it is not the default Workplace Forms Viewer response. Users accustomed to using the Viewer will be unfamiliar with this result.

When working in a multi-line field, the following cursor keys produce different results than in the Viewer:

### **Page Up**

Scrolls the text up one 'page'.

### **Page Down**

Scrolls the text down one 'page'.

### **CTRL + Page Up**

Moves the cursor to the beginning of the visible text.

### **CTRL + Page Down**

Moves the cursor to the end of the visible text.

## Printing

When a user prints a form through Webform Server, the server responds with a print preview that opens in a new window. The user can either print the form from the preview or close the preview and return to the original form.

The preview is a PNG image of the form that is generated on the server. This image is generated to a size that is determined by the server configuration. This means that the image is always generated to fit a specific page size, which in turn means that the page size cannot be changed from form to form or page to page.

Because the PNG image is generated on the server, any fonts used by the form must also reside on the server in order for the form to print correctly. In other words, when designing forms that users may print, make sure you use fonts that you can make available on the server.

## Signatures

Webform Server allows users to sign forms using Clickwrap signatures, and to verify other types of signatures that have already been applied to the form. However, Webform Server does not allow users to sign forms using any other signature types.

This means that signature-based security is limited when using Webform Server. While Clickwrap signatures prove acceptance of a document, they do not provide the same level of authentication as digital signatures.

Instead, the job of properly identifying the user is moved to the web application, which may use standard authentication measures for logging in users before they work with forms. See “Authenticate Users Through Common Practices” on page 9 for more details.

## Type Checking and Predictive Input Checking

Webform Server does not support predictive input checking. However, Webform Server does support type checking when the user exits a field.

For instance, if a field is set to accept phone numbers in the ###-#### format, the user will be able to enter any initial value into that field. However, once they shift focus to another item, the field will be highlighted as an error if the value does not match the template.

**Note:** Changing the default settings in the Webform Server *translator.properties* file (for example, setting **changeNotificationItems** to **none**) may result in type checking not working as described above. For detailed information about the *translator.properties* file, see the *IBM Workplace Forms Server — Webform Server Administrator’s Guide*.

## URLs

Webform Server does not allow users to submit forms to multiple URLs at the same time. Submissions are restricted to a single URL because of potential difficulties when updating the original XFDL form.

When Webform Server sends a form to the user, it keeps the original XFDL form on hand. When the HTML form is later submitted, Webform Server updates the XFDL form with the submitted data. However, if two sets of data were submitted, as they would be if you were submitting to two URLs, Webform Server would have difficulty synching both data sets with the original XFDL form. For this reason, multiple URLs are not supported.

**Note:** XFDL 7.0 does not support submissions to multiple URLs. Only older versions of XFDL support submissions to multiple URLs. In other words, you cannot setup an XFDL 7.0 form so the Viewer submits it to multiple URLs.

## Viewer Settings

While the Viewer supports a number of specialized settings through the *ufo\_settings* option, Webform Server ignores all Viewer Settings except for the *menu*, *printwithformaterrors*, *savewithformaterrors*, *signwithformaterrors*, and *submitwithformaterrors* settings.

Of those settings, all of them work exactly as they do in the Viewer, with the exception of the *menu* setting. This setting supports the toolbar buttons that are specific to Webform Server. The following table lists those buttons and provides the appropriate tag for each:

Icon	Description	Tag
Open	Opens a new form.	open

Icon	Description	Tag
Save	Save the current form.	save
Print	Prints the current form.	print
Refresh	Refreshes the current page.	refresh
Accessibility	Toggles accessibility mode on and off.	accessibility

## XForms

When creating a form using XForms, the XForms elements are enclosed or *skinned* within XFDL elements. As a result, Webform Server’s support of XForms is based on Webform Server’s support of the associated XFDL items and options. See the *XFDL Specification* for details on how XForms elements are skinned by XFDL elements. See “Appendix A: Supported XFDL Options” on page 53 for a complete list of the XFDL options Webform Server supports.

## XForms Submissions

Webform Server ignores URLs for submissions of type **replace=all**. In the Viewer, submitting to a URL submits the data to the URL. For example, you can submit data to another application, like a database, using a URL. With Webform Server, submitting to a URL submits the data to the (Webform Server) servlet. This means that if you want to submit data (using **replace=all**) to another application, you cannot use only a URL submit; you must also retrieve the information from the servlet and direct it to the desired application.

**Note:** To create a submission of type **replace=all** (the default setting), you must create it within a **DOMActivate** event.

## XML Data Model

By default, the XML Data Model is updated while the user is filling out the form. However, changing the default settings in the Webform Server *translator.properties* file (for example, setting **changeNotificationItems** to **none**) may result in the XML Data Model not being updated while the user is filling out the form. If so, the user must click an “update” button to update all computes in the form, including the XML Data Model. For detailed information about the *translator.properties* file, see the *IBM Workplace Forms Server — Webform Server Administrator’s Guide* for more details.

## Other Differences

The following table lists a number of other differences that do not require additional explanation. Additionally, you should review “Appendix A: Supported XFDL Options” on page 53 for a complete list of the XFDL options Webform Server supports.

Functionality	Webform Server	Viewer
Calendar Widget	Not supported — Webform Server converts the calendar widget to a date format field	Supported
e-mail	Partial support — Users must save forms to their local computer and e-mail them as attachments via e-mail program	Full support

<b>Functionality</b>	<b>Webform Server</b>	<b>Viewer</b>
Form version support	Version 6.0 and later	Versions 4.4 and later
Inactive cells (cells that have their <i>active</i> option set to <i>off</i> )	Internet Explorer — Inactive cells are omitted from popups, comboboxes, and lists.  Mozilla-based browsers — Inactive cells are displayed but are disabled.	Inactive cells are displayed but are disabled.
Rich Text Fields	Not supported — Webform Server converts rich text fields to plain text fields.	Supported
Schema	Server-side only	Client and Server
Screen readers	JAWS only	MSAA compliant
Slider	Not supported	Supported
Smartfill	Not supported	Supported
Spellchecking	Not supported	Supported
User modification of display or print preferences	Not supported	Supported
Viewer functions, such as <i>fileOpen</i> , <i>messageBox</i> , <i>setCursor</i> , and so on.	Not supported	Supported
Webservices	Supported in XForms forms only.	Supports webservices in both XFDL and XForms forms.
Zoom capability	Not supported	Supported
URIs starting with "file:"	Not supported	Supported

---

## General Best Practices

When designing forms for use with Webform Server, there are a number of practices that you should follow. These practices will ensure that the form operates properly both with Webform Server and with the Viewer. You should consider them regardless of whether you are designing new forms or modifying forms originally designed for the Viewer for use with Webform Server.

The following topics are discussed:

- “Authenticate Users Through Common Practices.”
- “Do Not Rely on the Size of a Field to Limit Input” on page 10.
- “Use Help Messages” on page 10.
- “Design Forms for Print to Use the Same Page Size” on page 11.
- “Compress All Forms” on page 12.
- “Portals: Do Not Use Cancel Buttons” on page 13.
- “Portals: Disable Unnecessary Toolbar Buttons” on page 13.
- “Do Not Use Rich Text” on page 14
- “Test Forms Thoroughly” on page 15.

---

## Authenticate Users Through Common Practices

Webform Server only supports Clickwrap signatures. While Clickwrap signatures provide proof of acceptance, they do little to prove the identity of the user. At best, they provide weak authentication. This restriction is normally overcome by using stronger signature technology, such as digital signatures. However, this is not possible with Webform Server.

Instead, if security is a concern, use common web authentication practices such as basic authentication.

### Examples

The simplest solution is to enable basic authentication through your web server or your processing application. These applications can authenticate users based on an ID and password before serving forms to them.

Other options include solutions such as NTLM, Kerberos, or SSL with Mutual Authentication.

### Exceptions to this Practice

Authenticating users is unnecessary if your forms do not include signatures, or if your forms application has a limited and trusted user-base. However, if security is a concern, you should authenticate all users as described.

---

## Do Not Rely on the Size of a Field to Limit Input

In some cases, form designers may rely on the size of a field to limit the amount of text the user can enter. For example, a multi-line field with a *scrollhoriz* of **wordwrap** and a *scrollvert* of **fixed** normally prevents the user from typing beyond the visible limit of the field.

However, Webform Server cannot restrict input in this manner. Instead, the following rules apply:

- **Single-Line Fields** — You can use the *format* option to limit the amount of text the user can type.
- **Multi-Line Fields** — There is no way to limit the amount of text the user can type.

### Example

The following field uses the length setting in the *format* option to limit text to 100 characters. Note that this assumes a single-line field.

```
<field sid="Description">
  <value></value>
  <format>
    <datatype>string</datatype>
    <constraints>
      <mandatory>optional</mandatory>
      <length>
        <min>0</min>
        <max>100</max>
      </length>
    </constraints>
  </format>
</field>
```

### Exceptions to This Practice

There are no exceptions to this practice.

---

## Use Help Messages

Although Webform Server can check user input for formatting errors when the user exits a field, unlike Viewer forms, Webform Server forms cannot predictively check user input for formatting errors as the user enters information within a field. If your database requires users to enter information in a particular format (such as dates in DDMMYYYY), you may want to use help messages to make this formatting information apparent to your users.

Help messages are popup messages that appear when users pass their mouse over a form item with help. You can use these help messages to explain the type of information required, or the type of formatting that is necessary. Additionally, using help messages to provide this information allows you to provide the assistance your users need without cluttering the form's appearance with multiple labels explaining the restrictions.

### Example

The following diagram shows an example of how you can use a help message to provide both formatting information and user help:

## Personal Information - page 7 of 10

How many dependents do you have?

Please type a two-digit number. For example, 02 would indicate two dependents.

In this case, dependents are defined as children you support, regardless of whether they live in your home, adult children under the age of 25 in a registered post-secondary institution, adult relatives with mental or physical disabilities that live in your home, or elderly relatives that live in your home.

NEXT >>

## Exceptions To This Practice

The hover help messages only appear for 5 seconds at a time. These messages should be kept as concise as possible so that they are easier for users to read. This caveat applies to Webform Server forms only; it does not apply to Viewer forms embedded inside a Webform Server application.

---

## Design Forms for Print to Use the Same Page Size

Unlike the Viewer, Webform Server does not print directly from the form on the screen. Instead, Webform Server generates a PNG image of the form. This image is generated to fit a particular size of page (the size of the page is set in the configuration options for Webform Server), and the size of the page cannot be changed from form to form or page to page.

This means that you should design all forms for print to fit the same page size. If you do not, the image of the form will be scaled to fit the page size, which may result in loss of detail.

### Example

When setting the page size for Webform Server, you must account for the margins imposed by the web browser. Internet Explorer automatically uses a 3/4" margin on all sides. This means that if you wanted to design your forms for an 8.5" x 11" page, you would set the page size in Webform Server to 7" x 9.5" (the page size minus the browser margins).

This setting is made in the *translator.properties* file, and would look like this:

```
printPageWidth = 7
printPageHeight = 9.5
```

For more information about configuring Webform Server, refer to the *Webform Server Administration Manual*.

## Exceptions To This Practice

You can disregard this practice if :

- Your users will never print the form.
- You have tested your form to ensure that printing from Webform Server does not cause a loss of detail or other visual problems.

---

## Compress All Forms

Webform Server will load and process forms more quickly if they are compressed. This will reduce CPU time required to process forms, which will in turn reduce the overall load on your server.

With this in mind, you should:

- Ensure that all of your forms are saved in a compressed format.
- Ensure that all of your forms are configured to submit in a compressed format.

### Example

#### To save a form in compressed format:

1. Load the form in the Designer.
2. In the **Outline** view, select **Form Global**.
3. In the **Properties** view, click the **Advanced** button.
  - The **Advanced** property appears in the **Properties** view.
4. Next, expand the **Advanced** property.
5. Next to **saveformat**, click the **value** field.
6. From the dropdown list, select **application/vnd.xfdl;content-encoding="base64-gzip"**.
7. Save the form.

The form will be saved in compressed format. All Workplace Forms products can open forms compressed in this manner, but you will not be able to edit the form in a text editor.

#### To configure a form to submit in compressed format:

1. Load the form in the Designer.
2. Select the submission button.
3. In the **Properties** view, click the **Advanced** button.
  - The **Advanced** property appears in the **Properties** view.
4. Next, expand the **Advanced** property.
5. Next to **saveformat**, click the **value** field.
6. From the dropdown list, select **application/vnd.xfdl;content-encoding="base64-gzip"**.
7. Save the form.

The form will now submit in compressed format when the user clicks the submission button. All Workplace Forms products can open forms compressed in this manner.

### Exceptions To This Practice

There are no exceptions to this practice.

---

## Portals: Do Not Use Cancel Buttons

Do not use buttons with a *type* of **cancel** in any forms that will be used in a portal environment. When used in a portal, cancel buttons close the entire browser window rather than just the form.

### Example

The following button has a *type* of **cancel**:

```
<button sid="Cancel">
  <type>cancel</type>
  <value>Cancel Form</value>
</button>
```

Using this sort of button in a portal environment will close the entire browser window rather than just the form.

### Exceptions To This Practice

There are no exceptions to this practice.

---

## Portals: Disable Unnecessary Toolbar Buttons

When running Webform Server in a portal environment, you may want to prevent users from saving and opening forms by disabling the buttons in the toolbar. In general, the philosophy of a portal is to provide a central location to work on forms. Allowing the user to save or load forms from their local computer may conflict with that philosophy, and may even cause unexpected problems with your workflow or other aspects of your portal application.

### Example

To disable the **save** and **open** buttons in the toolbar, you must use the *ufv\_settings* option. Place this option in the global page of the form, and use the *menu* setting to turn the **save** and **open** buttons off, as shown:

```
<globalpage sid="global">
  <global sid="global">
    <ufv_settings>
      <menu>
        <save>off</save>
        <open>off</open>
      </menu>
    <ufv_settings>
  </global>
</globalpage>
```

For more information using *ufv\_settings* to control the buttons in the toolbar, refer to "Viewer Settings" .

### Exceptions To This Practice

You can disregard this practice if your users need to work offline. In this case, you should test your application to ensure that users can load forms into your portal without any problems.

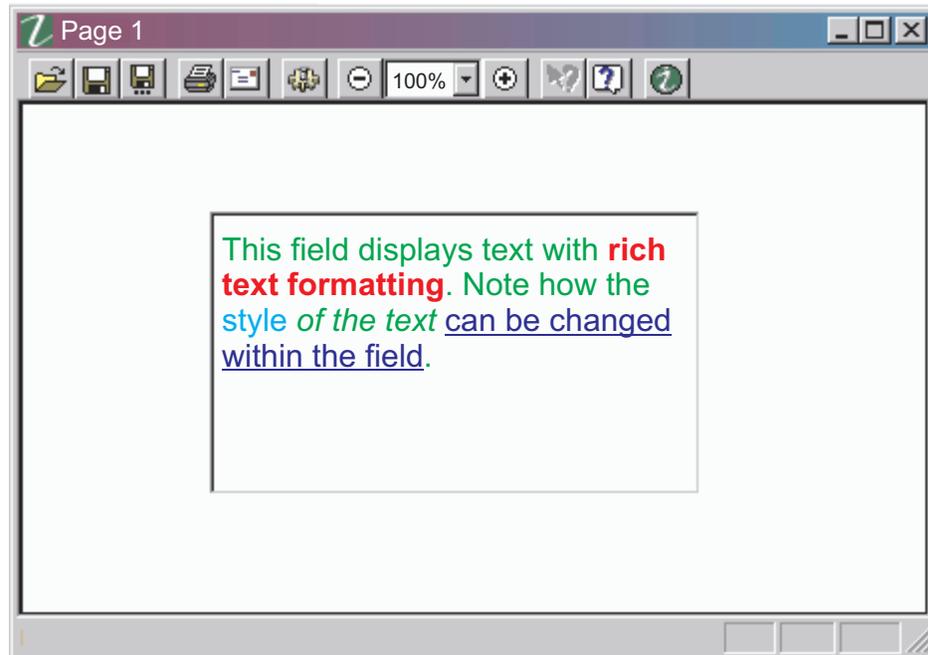
---

## Do Not Use Rich Text

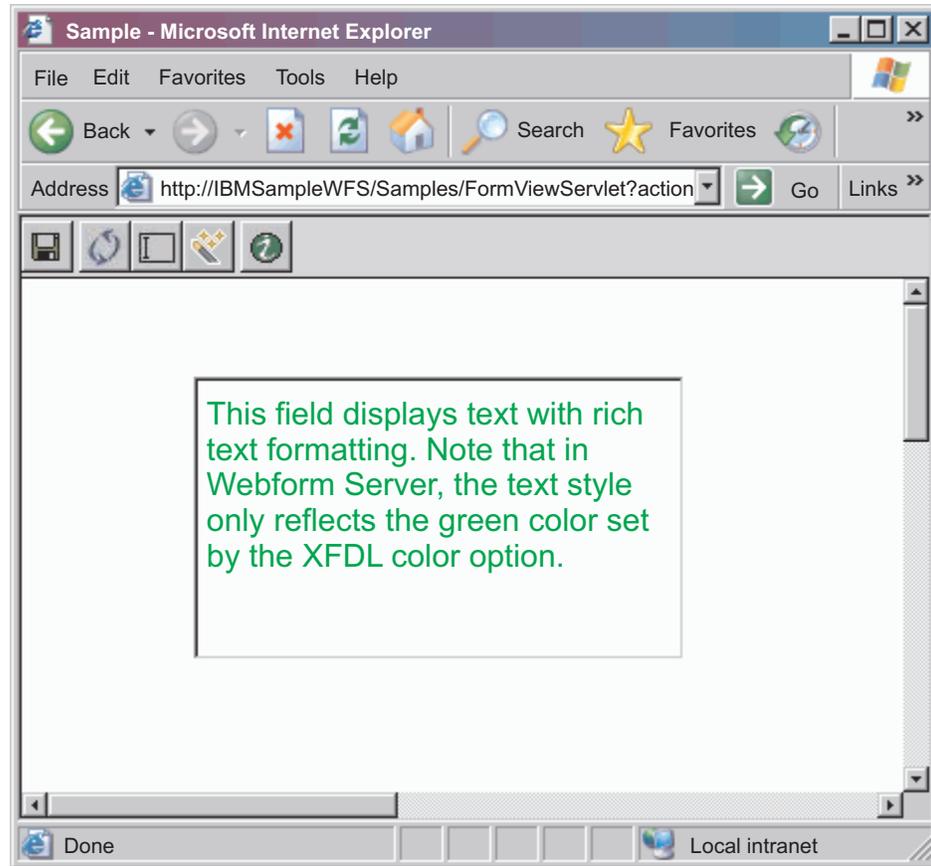
You should not use rich text formatted fields in Webform Server forms as most browsers cannot properly display the resulting rich text. Typically, any rich text formatting used to modify the field's text is stripped out, leaving only plain text. As a result, we recommend modifying text using XFDL options only.

### Examples

In the following example of a form in the Viewer, the green color is provided by the XFDL *color* option. The other text styles are provided by rtf:



This next sample shows a form in Webform Server. Note how the only text style that remains is the green color provided by the XFDL *color* option:



### Exceptions To This Practice

There are no exceptions to this practice.

---

## Test Forms Thoroughly

You should always test your forms thoroughly before releasing them to your users.

### Examples

Examples of possible tests include:

- Viewing and completing the form in Webform Server in all target environments (Windows and/or Mac) to ensure the correct appearance and function of the form.
- Viewing and completing the form in the Viewer to ensure the form works in all environments.
- Printing the form from Webform Server to ensure the page size is correct.
- Using Webform Server, view the form on client computer with a "clean" Windows/Mac OS install to ensure that font substitutions do not affect the appearance of the form.

### Exceptions To This Practice

There are no exceptions to this practice.



---

## Creating Dynamic Webform Server Forms

In general, Webform Server supports option changes via dynamic updates or automatic page refreshes. When the user interacts with a form (by changing the value of a item or, optionally, changing focus from one item to another) and the change resulting from that interaction affects the visual state of the form, then Webform Server either:

- dynamically changes the visual state of the form without refreshing the entire page, or
- changes the visual state of the form by automatically refreshing the entire page. (If *autoRefresh* is turned off in the *translator.properties* file, the form will not be automatically refreshed.)

When designing forms, make sure you understand which options are supported by dynamic updates versus full page refreshes. (See “Appendix A: Supported XFDL Options” on page 53 for details.)

The following topics are discussed:

- “Minimize the Number of Full Page Refreshes.”
- “Restrict Use of Action Items” on page 18..
- “Use Page Breaks for Dynamic Form Design” on page 19.
- “Do Not Toggle Computes Off Of Event Model Options” on page 20.
- “Toggle Computes off of the Triggeritem” on page 20.
- “Use Action Items to Trigger Computes Off Of a Page Flip” on page 22.

---

## Minimize the Number of Full Page Refreshes

Minimizing the number of full page refreshes will maximize the performance of your Webform Server application by reducing bandwidth use and load on the server. Furthermore, it will improve the user experience by reducing load times.

A full page refresh requires the form to be sent back to the server, refreshed, and returned to the user. While a single full page refresh may take milliseconds, large form pages, multiple users, or multiple page refresh triggers can considerably slow the process.

However, there are alternatives. Instead of using computes that rely on options that required full page refreshes, design your form so that computes rely on options that support dynamic updates. For example, to highlight a label when the user tabs into a field, you could use a compute either to change the label text to a different color or to change the label text to bold. Changes in font color are updated dynamically, but changes in font style (for example, plain, bold, and so on) require an automatic page refresh. Therefore, a change in font color will happen faster than a change in font style. For detailed information regarding which options support dynamic updating and which require a full page refresh, see “Appendix A: Supported XFDL Options” on page 53.

**Note:** You can configure Webform Server to turn off full page refreshes entirely. See *IBM Workplace Forms Server — Webform Server Administrator’s Guide* for details.

## Examples

The following compute changes the label text to a different color:

```
<field sid="First_Name">
  <value></value>
  <label>First Name</label>
  <labelfontinfo>
    <fontname>Arial</fontname>
    <size>8</size>
    <labelfontcolor compute="CHECK1.value == 'on' ? ('blue') &#xA;
      : ''"></labelfontcolor>
  </labelfontinfo>
</field>
```

## Exceptions to this Practice

There are no exceptions to this practice.

---

## Restrict Use of Action Items

Webform Server instructs the browser to automatically refresh a form whenever user input results in certain changes to the form (for example, layout changes). As a side effect of a refresh, action items that are set to occur once will actually occur each time the form is refreshed. This means that actions may be triggered when you do not want them to be. Furthermore, actions that are set to repeat will not work properly when the form is viewed in Internet Explorer.

As a result, you should restrict or eliminate the use of action items, and ensure that you test all actions thoroughly.

## Examples

The following action item opens an informational web page when the form is first opened:

```
<action sid="OpenInfo">
  <delay>
    <type>repeat</type>
    <interval>600</interval>
  </delay>
  <type>link</type>
  <url>http://www.ibm.com/sample/instructions.htm</url>
</action>
```

The web page is intended to provide the user with extra instructions that they should review once before completing the form. However, each time the form is refreshed (either automatically by Webform Server or manually by the user), the action will trigger and load the web page again. Clearly, this will become an annoyance for users and will prevent them from completing the form quickly and easily.

## Exceptions to this Practice

Action items are only recommended if you need to trigger some computes off of a page flip. For more information, see "Use Action Items to Trigger Computes Off Of a Page Flip" .

---

## Use Page Breaks for Dynamic Form Design

Webform Server instructs the browser to automatically refresh a form whenever user input results in certain changes to the form (for example, certain layout changes). One way to minimize the number of refreshes is to divide your form into pages.

For example, you might design a form in which one section changes completely depending on whether the user is single or married. If the layout change requires Webform Server to refresh the form, you may prefer to divide the form into pages to eliminate the need for a refresh. For example, on the first page of the form the user might enter some personal information and indicate whether they are married or single. Then the user clicks a button to flip to the next page. The content of the second page is based purely on their marital status, and is calculated during the page flip. In this case, single users skip page two entirely and are shown page three, while married users see page two, in which they must provide some information about their spouse and dependents.

By relying on page flips in this way, you can minimize the number of refreshes users experience.

### Example

The following pictures illustrate a form in which the spousal information becomes visible when the user selects the appropriate check box:

**Personal Information 2 of 5**

Is passport requested in spousal surname?  Yes  No

**Personal Information 2 of 5**

Is passport requested in spousal surname?  Yes  No

Year of Marriage

Surname of Spouse (See instruction No. 1)

The change in form layout may require Webform Server to refresh the form, requiring the user to wait a few seconds to see the modified form.

Instead, you can add a page flip to the first page, as shown:

<p><b>Personal Information 2 of 5</b></p> <p>Is passport requested in spousal surname? <input checked="" type="checkbox"/> Yes <input type="checkbox"/> No</p> <p style="text-align: right;"><b>Next &gt;&gt;</b></p>	<p><b>Spousal Information 1 of 1</b></p> <p>Year of Marriage  <input type="text"/></p> <p>Surname of Spouse (See instruction No. 1)  <input type="text"/></p>
---	---

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 calculation that produces the special section part of the normal flow of completing the form and provides the user with a more intuitive experience.

### Exceptions to this Practice

Alternately, you can design your form using options that Webform Server can dynamically update, like **visible** and **active**.

## Do Not Toggle Computes Off Of Event Model Options

### Examples

Do not toggle computes off of the *activated*, *dirtyflag*, *focused*, *focuseditem keypress*, or *mouseover* options. Computes based on these options do not work under Webform Server. Instead you should design your form so that computes are activated by changes to the form’s content. For example, toggling a visible/invisible section based on whether a particular checkbox has been selected.

No example provided

### Exceptions to this Practice

You may choose to enable computes based on the *focused* and *focuseditem* options by modifying the *focusNotificationItems* property in the *translator.properties* file. For more information about the *translator.properties* file, see the *IBM Workplace Forms Server - Webform Server Administrator’s Guide*.

## Toggle Computes off of the Triggeritem

Many form designs rely on computes that are triggered off of the *activated* option. For example, you may want certain computes to run when a button is clicked, so those computes are set to run when the button’s *activated* option goes from *off* to *on*.

Since Webform Server does not support the *activated* option, this strategy does not work. Clicking a button will not automatically trigger a compute. Instead, you must set your compute to toggle off of the *triggeritem* option, and set your buttons to a type of *refresh*.

When the user clicks a button of type *refresh*, Webform Server sends the form to the server for an update. It also sets the value of the form's *triggeritem* twice: first to nothing, and then to equal the scope identifier (SID) of the button that was pressed. This allows you to create a toggle that will run when the *triggeritem* changes from an empty string to the SID of the button. While the server is updating the form, any compute set to detect this change will run.

This means that instead of setting a compute to run when a button's *activated* changes from *off* to *on*, you set the same compute to run when the *triggeritem* changes from an empty string to the SID of that button.

## Example

The following code shows a button that is set to duplicate a row. In this case, the row contains only one field to simplify the example. Notice that the button is of type *select* and that the compute is set to toggle off of the *activated* option. This is standard for forms that run in the Viewer.

```
<button sid="addRowButton">
  <value>Add Row</value>
  <type>select</type>
  <custom:rowNumber>1</custom:rowNumber>
  <custom:totalRows>1</custom:totalRows>
  <custom:addRow xfdl:compute="(toggle(activated, 'off', &#xA;
    'on') == '1') &#xA;
    ? set('custom:rowNumber', custom:rowNumber + '1') &#xA;
    +. set ('custom:totalRows', custom:totalRows + '1') &#xA;
    +. set(duplicate('NameField_0', 'item', &#xA;
      'addRowButton', 'item', 'BEFORE_SIBLING', &#xA;
      'NameField_' + custom:rowNumber) + '.visible', &#xA;
      'on')"></custom:addRow>
</button>
```

In the next example, notice that the button's type has been changed to *refresh*, and the compute is now toggled off of the *triggeritem* option. Otherwise, the code is identical to the first example.

```
<button sid="addRowButton">
  <value>Add Row</value>\
  <type>refresh</type>
  <custom:rowNumber>1</custom:rowNumber>
  <custom:totalRows>1</custom:totalRows>
  <custom:addRow xfdl:compute="(toggle(
    global.global.triggeritem, '', &#xA;
    'PAGE1.addRowButton')== '1') &#xA;
    ? set('custom:rowNumber', custom:rowNumber + '1') &#xA;
    +. set ('custom:totalRows', custom:totalRows + '1') &#xA;
    +. set(duplicate('NameField_0', 'item', &#xA;
      'addRowButton', 'item', 'BEFORE_SIBLING', &#xA;
      'NameField_' + custom:rowNumber) + '.visible', &#xA;
      'on')"></custom:addRow>
</button>
```

Since the Viewer also respects the refresh type, this button will work in both Webform Server and the Viewer, duplicating the row when the user clicks the button.

## Exceptions to this Practice

If you need to trigger a compute off of a page flip, you cannot use the *triggeritem* to do this because the *pagedone* buttons do not set the *triggeritem*. Instead, you must use an action item to perform the page flip. For more information, see "Use Action

---

## Use Action Items to Trigger Computes Off Of a Page Flip

In some cases you may want to trigger a compute when the user clicks a page flip button. For example, before the Viewer changes the page, you may want to record which page the user is currently on so that you can return to it later.

To do this in the Viewer, you would normally toggle the compute off of the *activated* option of the *pagedone* button. However, because Webform Server does not support the *activated* option, this will not work. Furthermore, the Webform Server practice of toggling computes off of the *triggeritem* will not work either, because *pagedone* buttons do not set the *triggeritem*.

To work around this limitation, you must first change your button from a type of *pagedone* to a type of *refresh*. This will set the *triggeritem* when the button is clicked, which you can then use to trigger a compute.

However, the button will no longer flip the page. To make this happen, you must add an action item to the form that will perform the page flip when the user clicks the *refresh* button. To do this, you must set the *active* option of the action to *off*, and then toggle it to *on* whenever the *refresh* button is clicked. This will trigger the action, which will then flip the page.

This creates the following sequence of actions:

1. The user clicks the *refresh* button.
2. This sets the *triggeritem* option in the form, and causes the page to update.
3. Any compute set to toggle off the *triggeritem* is run.
  - This must include a compute that sets the *active* option of the action item to *on*.
4. The action is triggered, and flips to the appropriate page of the form.

Once the user has changed the page, the action item will not run again unless they return to that page. If you allow the user to return to the first page, then you must make sure the action is turned *off* again (usually through a compute that is triggered by flipping to that page).

### Example

In this example, when the user clicks the “summary page” button, a reference to the current page is set into a button on the summary page. This allows the user to return to the current page from the summary page. To create this effect, we must use a *refresh* button and a *pagedone* action, as shown:

```
<button sid="GoToSummary">
  <type>refresh</type>
</button>
<action sid="PageFlip">
  <active>off</active>
  <delay>
    <type>once</type>
    <interval>0</interval>
  </delay>
  <type>pagedone</type>
```

```

    <url>#Page4.global</url>
    <custom:actions xfdl:compute="your computes here"
      ></custom:actions>
  </action>

```

In this case, we need the compute to perform these actions:

1. Set the “PageFlip” action in the summary page to point to the first page.
  - This ensures that the user will return to the current page when they click the “Return” button.
2. Set a custom option in the summary page with the name of the referring page.
  - This ensures that you can turn the action item off in the referring page before flipping back to it.
3. Set the action item on the summary page to be inactive.
  - This ensures that the *pagedone* action is turned off, in case a previous page flip left it turned on.
4. Set the action item on the current page to be active.
  - This activates the action item, which flips the page.

Furthermore, the compute must be triggered by the user clicking the “PageFlip” button. The following compute demonstrates this, assuming that the summary page is Page4 and the custom option is called custom:referer and is part of the action item:

```

(toggle(global.global.triggeritem) == '1') and
(global.global.triggeritem == 'Page1.GoToSummary') ?
set('Page4.PageFlip.url[0]', '#Page1.global') +
set('Page4.PageFlip.custom:referer', 'Page1') +
set('Page4.PageFlip.active', 'off') +
set('active', 'on') : ''

```

The summary page must have a similar button and action item, as shown:

```

<button sid="ReturnButton">
  <type>refresh</type>
</button>
<action sid="PageFlip">
  <active>off</active>
  <delay>
    <type>once</type>
    <interval>0</interval>
  </delay>
  <type>pagedone</type>
  <url></url>
  <custom:actions xfdl:compute="your computes here"><
    /custom:actions>
</action>

```

However, the compute in the summary page does not need to do as much work. It simply needs to perform these actions:

1. Set the action item in the referring page to be inactive.
  - This ensures that the *pagedone* action is turned off, in case a previous page flip left it turned on.
2. Set the action item on the current page to be active.
  - This activates the action item, which flips the page.

Once again, the compute is triggered by the user clicking a button. In this case, the “Return” button. The following compute demonstrates this:

```
(toggle(global.global.triggeritem) == '1') and  
  (global.global.triggeritem == 'Page4.ReturnButton') ?  
  set(custom:referer + '.PageFlip.active', 'off') +  
  set('active', 'on') : ''
```

## Exceptions

There are no exceptions to this practice.

---

## Choosing the Right Fonts

Choosing the correct fonts when designing your forms will make the form completion process easier and more pleasurable for your users. In general, you should always choose fonts that are easily to read and widely available.

This section gives helpful suggestion on how to choose appropriate fonts for your forms. The following topics are discussed:

- “Use Only Fonts that are Installed on Client Computers”.
- “Do Not Use Symbol or Wingdings Fonts” on page 26
- “Use Matching Windows and Mac Fonts” on page 27
- “Recommended Japanese Fonts” on page 27

---

### Use Only Fonts that are Installed on Client Computers

When the form is displayed on the client computer, the web browser relies on the fonts installed on that computer. This means that if the appropriate font is not available, the web browser will substitute a “closest match” font. This can change the appearance of the form, and may actually cause some information to become obscured if the substituted font has a significantly different size.

If you cannot be sure which fonts are installed on client computers (for example, if the form is intended for the general public), you should restrict your form to fonts that are installed by default with the client computers (that is, fonts that are installed by default with Windows or Mac OS).

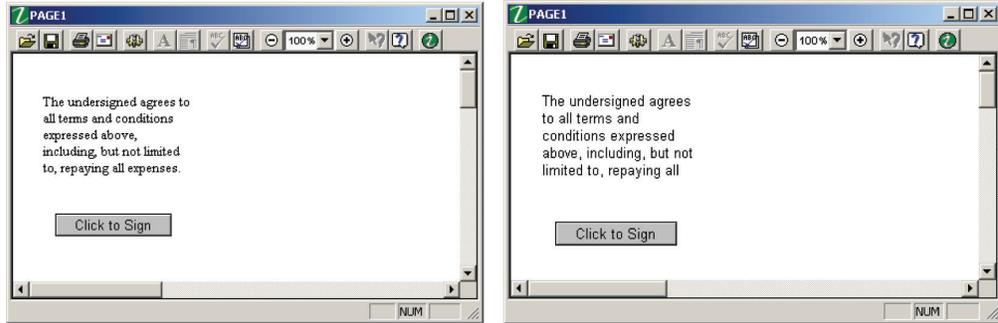
A similar problem may occur when you design a form using fonts available on a Windows system if the form is viewed on a Mac system. Even if there is a font available on both Windows and Mac systems having the same name, the actual font information may not be identical. This can result in differences in text appearance and layout. For example, text may be slightly wider on Mac systems, resulting in labels wrapping onto two lines.

When designing forms for use on Mac systems, make sure you use fonts that are identical on both Windows and Mac systems, not just fonts that have the same name, and test your forms on both Windows and Mac systems. Most importantly, test your form thoroughly.

International fonts are typically different on Windows and Mac systems. If you design a form using non-English text for use on a Mac system, test your form thoroughly.

#### Example

The following diagrams show two versions of the same form. The form on the left uses the Times New Roman font at a point size of 8. The form on the right uses the Arial font, also at a point size of 8. Notice how changing the font causes some information to disappear below the bottom of the label.



## Exceptions To This Practice

You can disregard this practice if:

- You provide the proper fonts to your users before they view your forms.
- You test your form on a “clean” Windows or Mac installation to ensure that any font substitutions made do not adversely affect the form.

---

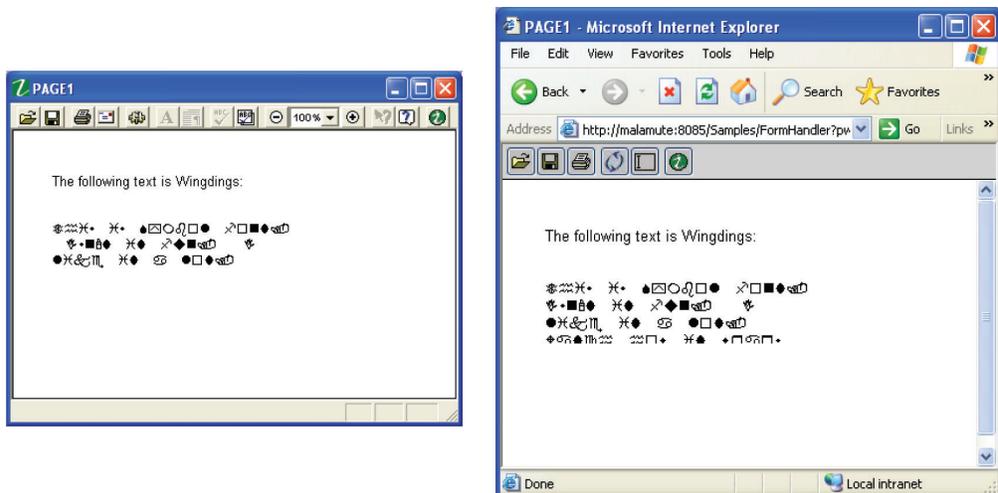
## Do Not Use Symbol or Wingdings Fonts

Webform Server does not properly render Symbol or Wingdings fonts. Not only do some of the symbols display incorrectly, but text and the items containing the text may be sized improperly as well.

If you need to include a symbol on your form that is normally contained within the Symbol or Wingdings fonts, we recommend that you use an image instead.

### Example

The following diagrams show a form that contains text in the Wingdings font. On the left, the form is displayed in the Viewer, while on the right, the form is displayed in Webform Server. Notice that while the Viewer displays only three lines of text, Webform Server displays four lines (although the fourth is cut off at the bottom). Furthermore, notice that the text wrapping is different. In the Viewer the form begins the second line with a space, while in Webform Server it does not.



## Exceptions To This Practice

There are no exceptions to this practice.

---

## Use Matching Windows and Mac Fonts

Fonts may not display the same way on a Mac system as they do on a Windows system. Even if there is a font available on both Windows and Mac systems having the same name, the actual font information may not be identical. This can result in differences in text appearance and layout. For example, text may be slightly wider on Mac systems, resulting in labels wrapping onto two lines.

When designing forms for use on Mac systems – especially when using international fonts – make sure you use fonts that are identical on both Windows and Mac systems, not just fonts that have the same name, and test your forms on both Windows and Mac systems. Most importantly, test your form thoroughly.

### Example

No example provided.

## Exceptions To This Practice

You can disregard this practice if:

- You provide the proper fonts to your users before they view your forms.
- You test your form on a “clean” Windows or Mac installation to ensure that any font substitutions made do not adversely affect the form.

---

## Recommended Japanese Fonts

You may find that forms that contain Japanese fonts do not print correctly. While this does not interfere with the presentation of forms online, it may negatively affect users who wish to print the form for their own records. To avoid this problem, IBM recommends that you choose fonts from the following list:

- MS Mincho
- MS PMincho
- MS Gothic
- MS PGothic
- MS UI Gothic

### Example

No example provided.

## Exceptions To This Practice

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 30.
- “Formatting E-Mail Addresses” on page 32.

---

## 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 ensuring 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.



---

# Designing Accessible Forms

---

## Why Create 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.

---

## Topics Discussed

This section describes practices you should follow whether you are designing new accessible forms or updating existing ones. The emphasis is on issues that apply to XFDL forms that will be read by Freedom Scientific's JAWS for Windows 4.0 screen reading software. It includes the following topics:

- "Provide Appropriate Accessibility Messages".
- "Put Label Text Into Aclabels".
- "Use Field Items To Display Text Information".
- "Place Graphics Inside Buttons".
- "Minimize and Explain the Use of Dynamic Content".
- "Reset the Form's Tab Order".
- "Identify Row and Column Headings".
- "Use Contrasting Page Background Colors".
- "Do Not Use List Items".
- "Test Forms Thoroughly".

---

## 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/](http://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/](http://www.microsoft.com/enable/) — Discusses Microsoft's built-in accessibility features.
- [www.access-board.gov/sec508/508standards.htm](http://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/](http://www-3.ibm.com/able/) — Product and service information for people with disabilities.
- [ncam.wgbh.org/webaccess/](http://ncam.wgbh.org/webaccess/) — Accessibility projects by the National Centre for Accessible Media.

---

## Provide Appropriate Accessibility Messages

Although JAWS recognizes most items that appear on a form, the default information it provides 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.

The *acclabel* option can help you meet those requirements. Using an *acclabel* allows you to provide appropriate messages to help users with disabilities without impacting users without impairments.

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

The following general practices can help you create useful messages:

- Repeat the text of labels in the *acclabel* of the following item.
- 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, JAWS reads an item's help message, so *acclabels* do not need to repeat that content. For more information on JAWS announcements and proper accessibility practices when designing forms, refer to Appendix B: JAWS Announcements.

### Example

When building an accessible form, it's important to be familiar with JAWS and the default phrases it announces for each item in addition to your custom accessibility message. Before continuing with this section, you may want to review Appendix B: JAWS Announcements

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 JAWS are italicized in the text.

## Field

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

A diagram showing a text input field. The label "First Name" is positioned above the field. The field itself is a rectangular box with a thin border and a light gray background.

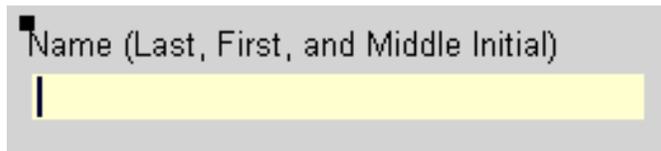
JAWS reads field messages in the following order:

- label option
- accessibility message
- the value, or empty value status
- JAWS message
- help message

**Note:** For details on JAWS messages and how they integrate with form messages, see Appendix B: JAWS Announcements.

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.

A diagram showing a mandatory text input field. The label "Name (Last, First, and Middle Initial)" is positioned above the field. The field itself is a rectangular box with a thin border and a yellow background, indicating it is mandatory. A small black square is located to the left of the label.

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 JAWS does 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.

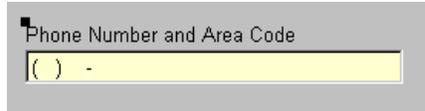
The following code shows a suitable accessibility message for this field:

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

As a result, when JAWS 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. Type in text.*"

**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:

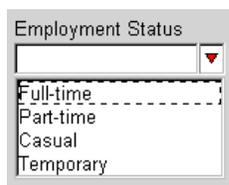
```
<acclabel>  
  Type your area code and phone number into this mandatory  
  field. You do not need to type the parenthesis around the  
  area code or the dash in the phone number. These symbols  
  have been provided for you.  
</acclabel>
```

When users tab to this field, JAWS announces: "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. Type in text.*"

You should note that JAWS pronounces most punctuation, including parentheses, number signs, and dashes that appear in labels, acclabels, and item contents. For example, JAWS 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. JAWS automatically provides instructions for using the combobox.

JAWS reads combobox messages in the following order:

- label option
- accessibility message
- the selected choice, or empty choice status
- JAWS message
- help message

**Note:** For details on JAWS messages and how they integrate with form messages, see "Appendix B: JAWS Announcements".

Because JAWS 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:

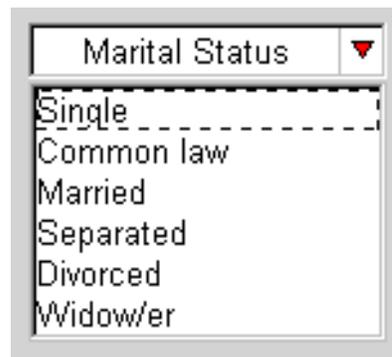
```
<acclabel>  
  Type your employment status or select a choice from the list.  
  This list contains 4 items. The Employment Status combobox.  
</acclabel>
```

When JAWS 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, JAWS reads the choices and their list status aloud. For example, "Full-time. 1 of 4."

### 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. JAWS automatically announces instructions for using popup lists. The following diagram shows a typical popup list:



JAWS reads popup messages in the following order:

- accessibility message
- label option or selected choice
- JAWS message
- help message

**Note:** For details on JAWS messages and how they integrate with form messages, see "Appendix B: JAWS Announcements".

Unlike comboboxes, JAWS reads 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, JAWS reads: "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, JAWS reads the choices and their list status aloud. For example, "Married. 3 of 6."

### Check Box

JAWS reads aloud both the checkbox's accessibility message and label. However, form designers frequently use a separate label item for checkboxes to allow more flexible placement options for the label. As a result, to make checkboxes accessible, you should repeat the contents of the label in the *acclabel* item. The following diagram shows a check box and its label:



JAWS reads check box messages in the following order:

- label option
- accessibility message
- value
- JAWS message
- help message

**Note:** For details on JAWS messages and how they integrate with form messages, see "Appendix B: JAWS Announcements".

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 checkbox:

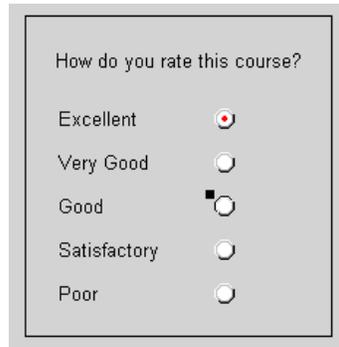
```
<acclabel>
  Select this checkbox if you would like an agent to contact
  you directly.
</acclabel>
```

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 checkbox, JAWS reads: "Would you like an agent to contact you directly? Select this checkbox if you would like an agent to contact you directly. *Checkbox: checked/not checked. To activate, press spacebar.*"

### Radio Button

As with checkboxes, form designers frequently use separate label items for radio buttons. As a result, 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:



JAWS reads radio button messages in the following order:

- label option
- accessibility message
- value
- JAWS message
- help message

**Note:** For details on JAWS messages and how they integrate with form messages, see "Appendix B: JAWS Announcements".

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 47 or "Put Label Text Into `Acclabels`".

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, JAWS would read: "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. However, JAWS 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. You should include this text in the *acclabel* of the appropriate input items. This allows JAWS to inform users of any changes or instructions.

**Note:** This practice is an alternative to "Use Field Items To Display Text Information"

### 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:

	YES	NO
<b>EYESIGHT</b>		
1. HAVE YOU LOST USE/SIGHT OF EITHER EYE?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
2. IS PERIPHERAL (SIDE) VISION RESTRICTED?	<input type="checkbox"/>	<input checked="" type="checkbox"/>

The following code creates an *acclabel* option for the "Yes" checkbox shown in the previous diagram. Note that the checkbox *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
    checkboxes. 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 checkbox.
  </acclabel>
</check>
```

When a JAWS 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 checkboxes."

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 checkbox. *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), JAWS typically reads:

- The *label option* of the item.
- The accessibility message.
- The contents of the item.
- Any instructions JAWS automatically adds for using a particular item type.

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*.

This practice may result in lengthy accessibility messages. In many cases, this 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 may be more useful to split the instructions across multiple items for a vision impaired user.

## 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".

---

## Use Field Items To Display Text Information

As an alternative to putting label text in *acclabel* options, 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 and borderless fields that receive the focus. When properly formatted in this way, fields can be virtually indistinguishable from labels. However, unlike labels, JAWS recognizes fields as normally accepting user input and therefore automatically reads any text they contain. However, 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.

**Note:** This practice is an alternative to "Put Label Text Into Acclabels".

## Example

When a field receives the focus (for example, by using the tab key to navigate to the field), JAWS 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:

Select Items for Purchase			
<i>Quantity</i>	<i>Select Product</i>	<i>Unit Price</i>	<i>Amount</i>
	--- Select --- ▼		\$0.00
	--- Select --- ▼		\$0.00

In this case, when JAWS users tab into a read-only field, they hear JAWS read 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 "Provide Appropriate Accessibility Messages" on page 37.

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, JAWS 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. Use your reading keys to read the text.*"

**Note:** If you use labels to identify other items, such as fields, checkboxes, 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 JAWS 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".

---

## **Assist JAWS Users to Enter Forms Mode**

JAWS does not automatically enter Forms mode when reading text in the Viewer, nor does it retain Forms mode when switching pages in the Viewer. This means that form input elements may behave differently for JAWS users than they do for other users. To ensure that screen reader users can easily and consistently complete your forms, you should make it as simple as possible for them to enter or return to Forms mode while they are using your forms.

To ensure that JAWS users have the same forms experience as sighted users, you should ensure that the first item on every page of a form is a field. Furthermore, you should add a note to the *acclabel* on this field to remind screen reader users to press Enter so that they enter Forms mode.

### **Example**

No example provided.

### **Exceptions To This Practice**

There are no exceptions to this practice.

---

## **Place Graphics Inside Buttons**

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, JAWS reads the accessibility message. If Viewer Help is turned on when users tab into the button, JAWS reads the help message aloud 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 PureEdge Solution's logo</acclabel>
  <help>HELP5</help>
  <fontcolor>
  <bgcolor>
</button>
```

This code creates the help message for the button:

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

**Note:** Remember, Viewer Help must be turned on before JAWS 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 vision 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 changes 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. 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.

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.

### 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.

	<i>Quantity</i>	<i>Select Product</i>	<i>Unit Price</i>	<i>Amount</i>
<i>Row 1</i>	2	Product 3 ▼	\$300.00	\$600.00
<i>Row 2</i>		--- Select --- ▼		\$0.00
<i>Row 3</i>	4	Product 1 ▼	\$100.00	\$400.00
<i>Row 4</i>		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

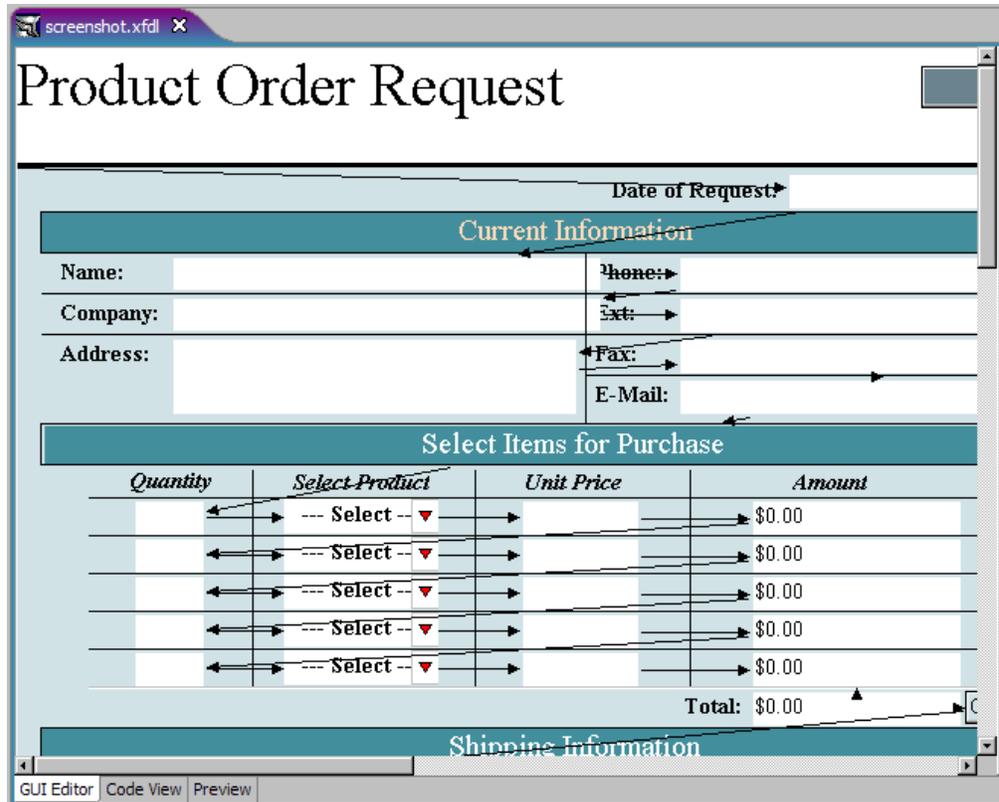
This practice is a reminder to carefully check and update your form's tab order.

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 dots indicate items that receive the input focus and 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.

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 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 JAWS 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.

Items for Purchase			
Quantity	Select Product	Unit Price	Amount
Row 1	--- Select ---		\$0.00
Row 2	--- Select ---		\$0.00
Row 3	--- Select ---		\$0.00
Row 4	--- Select ---		\$0.00
Row 5	--- Select ---		\$0.00
Total:			\$0.00

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, JAWS announces "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 bgcolor. 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 table lists the RGB triplet for these reserved colors:

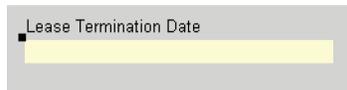
Reserved Color	RGB Triplet
"Mandatory" Yellow	255 255 208
"Invalid" Red	255 128 128
"Focus Indicator" Black	0 0 0

You should use a page background color that provides adequate contrast from these colors.

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.

### 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).



### Exceptions To This Practice

Currently there are no exceptions to this practice.

---

## Do Not Use List Items

You should avoid using *list* items in your forms. Instead, you should use comboboxes or popup lists. Form developers generally prefer comboboxes or popup lists anyway because they take up less space on the form. As a result, this practice will likely have no impact on the way you design your forms.

Although JAWS recognizes list items, it does not announce them as lists. JAWS simply reads the list label and accessibility message and directs users to use the arrow keys. If you cannot avoid using lists in your form, ensure that your label or accessibility message uniquely identifies each list so that users with visual disabilities can distinguish between multiple lists in the same form.

### Example

No example provided.

### 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. 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.

---

## Test Forms Thoroughly

You should always test your forms before releasing them for general use. You should test your forms using all of the accessibility software that a person with disabilities may use. This includes:

- Turn on your focus indicator.
- Use a screen magnifier.
- Run JAWS.
- 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.

Testing your forms using typical accessibility tools allows you to ensure that your form is fully accessible. This practice ensures that all visual elements in the form are represented by text that JAWS 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.



## Appendix A: Supported XFDL Options

The following table lists all XFDL options, and indicates how each option is supported by Webform Server. Webform Server still reads forms with unsupported options, but will not set these options or respect the settings in these options.

In general, Webform Server supports option changes via dynamic updates or automatic page refreshes. When the user interacts with a form (for example, by changing the value of a item, changing focus from one item to another, and so on) and the change resulting from that interaction affects the visual state of the form, then Webform Server either:

- dynamically changes the visual state of the form without refreshing the entire page, or
- changes the visual state of the form by automatically refreshing the entire page.

**Note:** This table represents the default configuration of Webform Server. See the *IBM Workplace Forms Server — Webform Server Administrator's Guide* for details on changing the default configuration.

Option	Updated Dynamically	Updated by Automatic Refresh	Not Supported
acclabel		Y	
activated			Y
active	Y		
bgcolor	Y <sup>2</sup>		
border		Y	
borderwidth		Y	
boundingbox		Y	
colorinfo		Y	
columnwidth		Y	
coordinates	Y		
data		Y	
datagroup	Y		
delay		Y	
dirtyflag			Y
editstate		Y	
excludedmetadata	Y		
filename	Y		
first		Y	
focused			Y <sup>6</sup>
focuseditem			Y <sup>6</sup>
fontcolor	Y <sup>2</sup>		
fontinfo		Y	
format	Y <sup>1</sup>		
formid	Y		

Option	Updated Dynamically	Updated by Automatic Refresh	Not Supported
fullname	Y		
group		Y	
help		Y	
image		Y	
imagemode			Y <sup>3</sup>
itemfirst		Y	
itemlast		Y	
itemlocation		Y	
itemnext		Y	
itemprevious		Y	
itemtag		Y	
justify		Y	
keypress			Y
label	Y <sup>1 2</sup>		
labelbgcolor	Y <sup>2</sup>		
labelborder		Y	
labelbordercolor		Y	
labelborderwidth		Y	
labelfontcolor	Y <sup>2</sup>		
labelfontinfo		Y	
last		Y	
layoutinfo			Y
linespacing		Y <sup>4</sup>	
mimedata		Y	
mimetype		Y	
mouseover			Y
next		Y	
padding		Y	
pagefirst		Y	
pageid	Y		
pagelast		Y	
pagenext	Y		
pageprevious		Y	
previous		Y	
printbgcolor	Y		
printfontcolor	Y		
printing			Y
printlabelbgcolor	Y		
printlabelfontcolor	Y		
printsettings	Y <sup>5</sup>		

Option	Updated Dynamically	Updated by Automatic Refresh	Not Supported
printvisible	Y		
readonly		Y	
rtf			Y
requirements			Y
rowspacing		Y	
saveformat	Y		
scrollhoriz		Y	
scrollvert		Y	
signature	Y		
signatureimage	Y		
signdatagroups	Y		
signdetails			Y
signer	Y		
signformat	Y		
signgroups	Y		
signitemrefs	Y		
signitems	Y		
signnamespaces	Y		
signoptionrefs	Y		
signoptions	Y		
signpagerefs	Y		
size		Y	
suppresslabel		Y	
texttype			Y
thickness		Y	
transmitdatagroups	Y		
transmitformat	Y		
transmitgroups	Y		
transmititemrefs	Y		
transmititems	Y		
transmitnamespaces	Y		
transmitoptionrefs	Y		
transmitoptions	Y		
transmitpagerefs	Y		
triggeritem	Y		
type		Y	
url	Y		
value	Y <sup>1</sup>		
visible	Y		
visiblerows		Y	

Option	Updated Dynamically	Updated by Automatic Refresh	Not Supported
webservices			Y
writeonly		Y	

<sup>1</sup>If the value change is meant to result in the item being implicitly resized, the resize is not supported by Webform Server. When designing your form, make sure you size the item so it will support the full range of values that users may enter.

<sup>2</sup>Changes to an option in a page or form global item will not be updated dynamically but will be updated by an automatic page refresh.

<sup>3</sup>The *imagemode* option only respects the default setting, which is **resize**.

<sup>4</sup>Setting the *linespacing* option too small will result in text overlapping. Make sure to test your form if you are using this option.

<sup>5</sup>The *printsettings* option only allows you to set which pages are printed. It does respect dialog or border settings.

<sup>6</sup>By default, the *focused* and *focuseditem* options are not supported. Focus changes are not posted to the server, and you cannot use computations to dynamically set the focus. If you modify the *translator.properties* file (by setting **focusNotificationItems** to **all**), the form will post focus changes to the server; however, this will result in very high network traffic.

## Appendix B: JAWS Announcements

This appendix lists the text that JAWS reads aloud for each form item. Knowing this information will help you create more meaningful accessibility messages for users who rely on JAWS. The list below uses the following placeholders 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 the list, defined by a *cell* item.

*<field 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.

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

*n* Indicates the number of choices in a list.

*<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 table shows the text that JAWS announces for each type of interactive form item in response to different user actions.

XFDL Item	User Action	JAWS Announces
Field (empty read/write)	Move to item using tab key. Move to item using SHIFT + TAB.	<i>&lt;label&gt;&lt;acclabel&gt;</i> Edit field is empty. Type in text.
Field (read/write with contents)	Move to item using tab key. Move to item using SHIFT + TAB.	<i>&lt;label&gt;&lt;acclabel&gt;</i> Edit field contains <i>&lt;field contents&gt;</i> . Type in text. <i>&lt;help&gt;</i>
Field (read-only containing text)	Move to item using tab key. Move to item using SHIFT + TAB.	<i>&lt;acclabel&gt;</i> Edit field contains <i>&lt;field contents&gt;</i> . Read-only. Use your reading keys to read the text. <i>&lt;help&gt;</i>
List (no selection)	Move to item using tab key. Move to item using SHIFT + TAB.	<i>&lt;label&gt;&lt;acclabel&gt;</i> Nothing selected. To move to an item, press the arrow keys. <i>&lt;help&gt;</i>
List (with selection)	Move to item using tab key. Move to item using SHIFT + TAB.	<i>&lt;label&gt;&lt;acclabel&gt;&lt;choice&gt;</i> selected. To move to an item, press the arrow keys. <i>&lt;help&gt;</i>
List (with or without selection)	Use up or down arrow to scroll through choices in list.	<i>&lt;choice&gt;</i> <i>x</i> of <i>n</i>

XFDL Item	User Action	JAWS Announces
Popup (empty)	Move to item using tab key. Move to item using SHIFT + TAB.	<acclabel><label> This is a popup list. Use the spacebar or down arrow key to bring up the list. To activate, press spacebar.<help>
Popup (after activating list)	Press the spacebar or down arrow key.	To move to an item, press the arrow keys.<help>
Popup (moving through list)	Press the arrow keys.	<choice> n of x
Popup (with selection)	Move to item using tab key. Move to item using SHIFT + TAB.	<acclabel> <choice> selected. This is a popup list. Use the spacebar or down arrow key to bring up the list. To activate, press spacebar.<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. Type in text.<help>
Combobox (after activating list)	Press the spacebar or down arrow key.	To move to an item, press the arrow keys.
Combobox (moving through list)	Use up or down arrow to scroll through choices in list.	<choice> n of x
Combobox (with selection)	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.<help>
Check (not selected)	Move to item using tab key. Move to item using SHIFT + TAB	<label><acclabel> Checkbox not checked. To activate, press spacebar.<help>
Check (selected)	Move to item using tab key. Move to item using SHIFT + TAB	<label><acclabel> Checkbox. Checked. To activate, press spacebar.<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. To activate, press spacebar.<help>
Radio (selected)	Move to item using tab key. Move to item using SHIFT + TAB.	<label><acclabel> Radio button. x of n. Selected. To activate, press spacebar.<help>
Button	Move to item using tab key. Move to item using SHIFT + TAB.	<label> Button <acclabel> To activate, press spacebar.<help>

---

## Appendix C: Additional Usage Notes

This appendix lists additional minor form design issues for Webform Server:

- If you draw a line that overlaps a combobox or popup, the Viewer will show the line on top. However, Webform Server will show the combobox or popup on top, which will cause a portion of the line to disappear. To address this problem, ensure lines do not overlap combo boxes or popups.
- Scroll bars may appear on fields despite the *scrollhoriz* and *scrollvert* settings. This occurs when one of the scroll bars is set to appear but the other is not. If the user types beyond the constraints of the field, the other scroll bar will appear despite the settings.
- Webform Server does not respect the *borderwidth* setting in the *printsettings* option. This means that printed forms will never display a border.
- If you use the *set* function, the value you are setting must precede the *set* function itself in the form's build order. For example, if a compute in Item A sets a value in Item B, then Item B must come before Item A in the build order. If the value you are setting follows the *set* function in the form's build order, the value will not be updated properly.

Note that using the *set* function to set a value in the same item will always work, regardless of build order.

- Submission buttons that are set to transmit the XML model do not work. No information is submitted.
- Action items that do not have a delay specified will not run. To correct this, add the *delay* option to the item.
- The *datagroup* option should always include the page reference. This allows forms to be fully compatible with both the Viewer and Webform Server. For example: `<datagroup>PAGE1.group1</datagroup>`
- The page reference in the *datagroup* option determines which page an attachment is associated with. Forms should be designed to manage all attachments from a single page.

Note that attachments added by users are included in these data groups and are associated with the same page reference.

- Impact Bold fonts that are 48 points or larger will render slightly wider than normal. This may cause lines of text to wrap incorrectly.
- When using *xforms:switch/xforms:case* in a form that will be used with Webform Server, the *xforms:switch* element must define and use the *xfdl:state* attribute. The *xfdl:state* attribute must define a reference that will be used to store the current state of the *xforms:switch* in an XForms model. If not used, the behavior of the *xforms:switch/xforms:case* is undefined. For more information about the state attribute, see *xforms:switch* in the *XFDL Specification*.
- The fonts included in the default Windows East Asian Language Pack do not contain any kerning information. For the most consistent and readable text, use a 12 point font.



---

## 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: 5724-N08

Printed in USA

S325-2596-00

