IBM® Workplace Forms™

Version 2.6.1

Using the Viewer Settings

S325-2610-00

Note
Before using this information and the product it supports, read the information in "Notices," on page 13.

First Edition (September 2006)

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

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

Contents

Using the Viewer Settings

This document describes the *ufv_settings* option. *ufv_settings* is an option that can be placed in a form to control some of the Viewer's features, such as:

- The information displayed in the Viewer's about box.
- The color used to denote errors on the form.
- The appearance of the cursor when the Viewer is in help mode.
- The color used to denote mandatory fields.
- The buttons that appear in the Viewer's toolbar.

To understand all of the controls available, you should review the beginning portion of each setting in this document.

About the ufv_settings Option

The *ufv_settings* option is declared in either the form global or any page global. As with other options, the global page settings override the global form settings.

The *ufv_settings* option belongs to the XFDL namespace. The option can control one or more features, and follows this syntax:

```
<ufv_settings>
   <feature1>
      <setting>setting_value1</setting>
      <setting>setting_valuen</setting>
   </feature1>
   <featuren>
      <setting>setting_value1</setting>
      <setting>setting_valuen</setting>
   </featuren>
</ufv_settings>
```

Note: Page settings override the global settings. For example, if you set the **validoverlap** globally, then set the **errorcolor** for page one, page one will not inherit the **validoverlap** setting. If you want to add page specific settings to your form, you must repeat the form's global settings in the settings for that page.

Usage Details

All *ufv_settings* can be changed by computes in the form. The following example shows how a form can be set to read-only using a compute. In this example, the custom option *opt_1* will set modifiable to **off** once the form has been submitted.

```
<button sid="SubmitForm">
   <value>Submit Form</value>
   <type>submit</type>
   <custom:opt_1 xfdl:compute="toggle(activated,'off','on') == &#xA;
      '1' ? set('global.global.ufv_settings[modifiable]', &#xA;
      'off'):''"></opt_1>
</button>
```

Note that the *ufv_settings* option is a form global option in the XFDL namespace, and is referenced using the following syntax:

```
global.global.ufv_settings[element][setting]
```

For example, to reference the setting for the Open control bar button, you use:

global.global.ufv_settings[*menu*][*open*]

To reference the setting for modifiable, you use:

global.global.ufv_settings[*modifiable*]

errorcolor

This feature sets the background color of an input item if a user's entry is incorrect (for example, if a user enters text in an integer field). The setting can use either RGB values, hex values, or a valid color name. See the *XFDL Specification* for a list of valid color names.

Example

```
<ufv_settings>
   <errorcolor>255,105,180</errorcolor>
</ufv_settings>
```

OR

```
<ufv_settings>
   <errorcolor>HotPink</errorcolor>
</ufv_settings>
```

OR

```
<ufv_settings>
   <errorcolor>#EE6AA7</errorcolor>
</ufv_settings>
```

All examples are correct. In each case, if an input item in the form receives an invalid entry, the background color of the item will change to hot pink.

Default

The default value for *errorcolor* is watermelon (255, 128, 128).

helpcursor

This feature controls the appearance of the cursor when the Viewer is in help mode. Valid settings are:

- **arrow** — The help cursor appears as the "normal select" cursor, as defined by the Windows® settings. This is normally a plain arrow.
- **question** — The help cursor appears as the "help select" cursor, as defined by the Windows settings. This is normally an arrow with a question mark beside it.

Note that the appearance of the cursor relies on the Windows settings. Individual users can use those settings to change the appearance of the cursor.

Example

```
<ufv_settings>
   <helpcursor>arrow</helpcursor>
</ufv_settings>
```

In this case, the help cursor is set to be an arrow rather than an arrow with a question mark.

Default

The default value for *helpcursor* is **question**.

mandatorycolor

This feature sets the background color of a mandatory input item. The setting can use either RGB values, hex values, or a valid color name. See the *XFDL Specification* for a list of valid color names.

Example

```
<ufv_settings>
   <mandatorycolor>255,228,225</mandatorycolor>
</ufv_settings>
```

OR

```
<ufv_settings>
   <mandatorycolor>misty rose</mandatorycolor>
</ufv_settings>
```

OR

```
<ufv_settings>
   <mandatorycolor>#FF00CC</mandatorycolor>
</ufv_settings>
```

All of the above examples are correct. In each case, the mandatory input items on the form will have a background color of misty rose. This background color will override any other background color stated in the form description, and cannot itself be overridden by a compute.

Default

The default value for *mandatorycolor* is light yellow (255, 255, 208).

menu

This feature allows a form designer to either hide or gray out the icons in the Viewer's toolbar or the toolbar used by Webform Server. The following table lists each icon, describes what the icon does, and provides the keyword to use with the menu setting.

Note that the Save Form and Save As button are controlled by the save keyword. This means that they must both either be on or off - you cannot set them individually.

Icon	Description	Keyword
Open	Opens a new form.	open
Save Form	Saves the form to the current file.	save
Save As	Saves the form, prompting the user for a filename and location.	save
Print	Prints the current form.	print
Mail	Emails the current form.	mail
Preferences	Opens the Preferences form.	preferences

Icon	Description	Keyword
Font	Opens the Font dialog, which allows the user to change font characteristics in rich text fields.	fontdialog
Paragraph	Opens the Paragraph dialog, which allows the user to change indenting and alignment in rich text fields.	paragraphdialog
Check Spelling	Checks the spelling in the current item.	spellcheck
Check All Spelling	Checks the spelling for the entire page.	spellcheckall
Zoom Out	Decreases the magnification of the form.	zoom
Select Zoom Factor	Sets the magnification to a specific factor.	zoom
Zoom In	Increases the magnification of the form.	zoom
Help	Activates the form's help messages.	help
Viewer Help	Opens the Viewer Help.	viewerhelp
Refresh Form	Refreshes the form, which updates all the computes. This feature is only available when viewing forms in Webform Server.	refresh
Toggle Accessibility Mode	Turns the accessibility mode on and off. This feature is only available when viewing forms in Webform Server.	accessibility

You can set each icon individually to one of three states:

- **on** — The icon is available as normal. This is the default setting.
- **off** — The icon is visible, but is grayed out and cannot be used.
- **hidden** — The icon is not visible. The toolbar will collapse appropriately to eliminate the gaps in the icons.

Example

```
<ufv_settings>
   <menu>
      <save>off</save>
      <print>on</print>
      <open>off</open>
      <mail>off</mail>
      <preferences>on</preferences>
      <spellcheck>on</spellcheck>
      <spellcheckall>on</spellcheckall>
      <help>hidden</help>
      <viewerhelp>on</viewerhelp>
   </menu>
</ufv_settings>
```

In a form with this *ufv_settings* declaration, the user will be able to select Print, Preferences, Check Spelling, Check All Spelling, and open the Viewer Help form. He or she will not be able to save the form, open another form, send mail, or access the help mode. Furthermore, the help mode icon will not be visible.

Note: The **menu** function does not prevent the form designer from adding controls for opening or saving forms, or from using other functions elsewhere in the form.

Default

All toolbar icons are **on** by default.

modifiable

This feature allows the read/write status of the entire form to be set.

Example

```
<ufv_settings>
   <modifiable>off</modifiable>
</ufv_settings>
```

A form with this setting would be read-only. This is not a secure method of preventing a form from being altered, since the source code can still be changed. However, this can be used to prevent accidental changes.

Default

The default value for *modifiable* is **on**.

printwithformaterrors

This feature allows form designers to create forms that can be printed even if there are type checking errors in the form. For example, you can create a form that the user cannot print unless they have completed all of the mandatory fields.

This element has three possible settings:

- **permit** — Allows prints to proceed even if there are type checking errors.
- **warn** — If type checking errors exist, warn the user and ask if they want to print the form anyway. This is the default value.
- **deny** — If type checking errors exist, alert the user and abort the print.

Note: This setting does not effect predictive type checking, which will still be applied as the user types.

To apply *printformaterrors* to all of the print buttons in your form, place the setting in the form global. For example:

```
<globalpage sid="global">
   <global sid="global">
      <ufv_settings>
         <printwithformaterrors>deny</printwithformaterrors>
      </ufv_settings>
   </global>
</globalpage>
```

To apply *printformaterrors* to all the print buttons on a single page, place it in the page global. For example:

```
<page sid="PAGE1">
   <global sid="global">
      <ufv_settings>
         <printwithformaterrors>deny</printwithformaterrors>
      </ufv_settings>
   </global>
   ...XFDL items...
</page>
```

If you want *printwithformaterrors* to apply to only a single button, you must implement it in a compute that references the print button you want to affect. The following example shows a compute that toggles between *permit* and *deny* based on the activated state of a print button. This ensures that users are denied the ability to print a form that contains format errors unless they are using the specified print button.

```
<ufv_settings>
   <printwithformaterrors compute="toggle( &#xA;
      page1.printButton.activated,'off', 'on') == '1' ? 'permit' : &#xA;
      'deny'"></printwithformaterrors>
</ufv_settings>
```

A form with this setting will deny printing until the **page1.printButton** is clicked. When that button is clicked, the form will toggle to permit and will print even if there are type checking errors. Once printing is complete, the setting will toggle back to *deny*.

Default

The default for *printwithformaterrors* is **warn**.

savewithformaterrors

This feature allows form designers to create forms that can be saved even if there are type checking errors in the form. For example, you can create a form that the user cannot save unless they have completed all of the mandatory fields.

This element has three possible settings:

- **permit** — Allows saves to proceed even if there are type checking errors.
- **warn** — If type checking errors exist, warn the user and ask if they want to save the form anyway. This is the default value.
- **deny** — If type checking errors exist, alert the user and abort the save.

  Note: This setting does not effect predictive type checking, which will still be applied as the user types.

To apply *savewithformaterrors* to all of the save buttons in your form, place the setting in the form global. For example:

```
<globalpage sid="global">
   <global sid="global">
      <ufv_settings>
         <savewithformaterrors>deny</savewithformaterrors>
      </ufv_settings>
   </global>
</globalpage>
```

To apply *savewithformaterrors* to all the save buttons on a single page, place it in the page global. For example:

```
<page sid="PAGE1">
   <global sid="global">
      <ufv_settings>
         <savewithformaterrors>deny</savewithformaterrors>
      </ufv_settings>
   </global>
   ...XFDL items...
</page>
```

If you want *savewithformaterrors* to apply to only a single button, you must implement it in a compute that references the save button you want to affect. The following example shows a compute that toggles between *permit* and *deny* based on the activated state of a save button. This ensures that users are denied the ability to save a form that contains format errors unless they are using the specified save button.

```
<ufv_settings>
   <savewithformaterrors compute="toggle( &#xA;
      page1.saveButton.activated,'off', 'on') == '1' ? 'permit' : &#xA;
      'deny'"></savewithformaterrors>
</ufv_settings>
```

A form with this setting will deny saving until the **page1.saveButton** is clicked. When that button is clicked, the form will toggle to permit and will save even if there are type checking errors. Once saving is complete, the setting will toggle back to *deny*.

Default

The default for *savewithformaterrors* is **warn**.

scrollfieldsonzoom

In some cases, using the Viewer's zoom feature can cause the text in fixed height fields to wrap incorrectly. This causes some of the text to disappear beyond the bottom of the field, which can prevent users from reading the text or entering as much text as they should be able to. To correct this problem, the Viewer adds scroll bars to fixed height fields when necessary.

You can disable this feature by using the *scrollfieldsonzoom* element. This element has two possible settings:

- **On** — Add scroll bars to fixed height fields when necessary.
- **Off** — Never add scroll bars to fixed height fields.

Disabling this feature may limit the functionality of the form when the user zooms.

Note: This feature overrides any settings in the Viewer's preferences form. Use this feature only in the form global settings. It does not work in the page global settings.

Example

The following example turns the scroll bars off:

```
<ufv_settings>
   <scrollfieldsonzoom>off</scrollfieldsonzoom>
</ufv_settings>
```

Default

By default, scroll bars are added to fixed height fields when necessary.

signwithformaterrors

This feature allows form designers to create forms that can be signed even if there are type checking errors in the form. For example, you can create a form that the user can sign even if they have not completed all of the mandatory fields.

This element has three possible settings:

- **permit** — Allows signing to proceed even if there are type checking errors.
- **warn** — If type checking errors exist, warn the user and ask if they want to sign the form anyway. This is the default value.
- **deny** — If type checking errors exist, alert the user and abort the signature.

Note: This setting does not effect predictive type checking, which will still be applied as the user types.

To apply *signwithformaterrors* to all of the sign buttons in your form, place the setting in the form global. For example:

```
<globalpage sid="global">
   <global sid="global">
      <ufv_settings>
         <signwithformaterrors>deny</signwithformaterrors>
      </ufv_settings>
   </global>
</globalpage>
```

To apply *signwithformaterrors* to all the sign buttons on a single page, place it in the page global. For example:

```
<page sid="PAGE1">
   <global sid="global">
      <ufv_settings>
         <signwithformaterrors>deny</signwithformaterrors>
      </ufv_settings>
   </global>
   ...XFDL items...
</page>
```

If you want *savewithformaterrors* to apply to only a single button, you must implement it in a compute that references the signature button you want to affect. The following example shows a compute that toggles between *warn* and *deny* based on the activated state of a signature button. This ensures that users are denied the ability to sign a form that contains format errors unless they are using the specified signature button.

```
<ufv_settings>
   <signwithformaterrors compute="toggle( &#xA;
      page1.saveButton.activated,'off', 'on') == '1' ? 'warn' : &#xA;
      'deny'"></signwithformaterrors>
</ufv_settings>
```

A form with this setting will deny signing until the **page1.signButton** is clicked. When that button is clicked, the form will toggle to warn and will alert users to formatting errors, but still allow them to sign the form. Once the form is signed, the setting will toggle back to *deny* **if** the ufv_setting has not been signed. To exclude this compute from the signature button, you must omit it from the signature. One way to do this is to omit *ufv_setting* from the *signoptions* filter in the signature button. For example:

```
<signoptions>
   <filter>omit</filter>
   <optionrefs>ufv_settings</optionrefs>
</signoptions>
```

Default

The default for *signwithformaterrors* is **warn**.

submitwithformaterrors

This feature allows form designers to create forms that can be submitted even if there are type checking errors in the form. For example, you can create a form that the user can submit even if they have not completed all of the mandatory fields.

This element has three possible settings:

- **permit** — Allows submits to proceed even if there are type checking errors.
- **warn** — If type checking errors exist, warn the user and ask if they want to submit the form anyway.
- **deny** — If type checking errors exist, alert the user and abort the submission. This is the default value.

Note: This setting does not effect predictive type checking, which will still be applied as the user types.

To apply *submitformaterrors* to all of the submit buttons in your form, place the setting in the form global. For example:

```
<globalpage sid="global">
   <global sid="global">
      <ufv_settings>
         <submitwithformaterrors>deny</submitwithformaterrors>
      </ufv_settings>
   </global>
</globalpage>
```

To apply *submitformaterrors* to all the submit buttons on a single page, place it in the page global. For example:

```
<page sid="PAGE1">
   <global sid="global">
      <ufv_settings>
         <submitwithformaterrors>deny</submitwithformaterrors>
      </ufv_settings>
   </global>
   ...XFDL items...
</page>
```

If you want *savewithformaterrors* to apply to only a single button, you must implement it in a compute that references the submit button you want to affect. The following example shows a compute that toggles between *permit* and *deny* based on the activated state of a submit button. This ensures that users are denied the ability to submit a form that contains format errors unless they are using the specified submit button.

```
<ufv_settings>
   <submitwithformaterrors compute="toggle( &#xA;
      page1.saveButton.activated,'off', 'on') == '1' ? 'permit' : &#xA;
      'deny'"></submitwithformaterrors>
</ufv_settings>
```

A form with this setting will deny submitting until the **page1.submitButton** is clicked. When that button is clicked, the form will toggle to permit and will submit even if there are type checking errors. Once submitting is complete, the setting will toggle back to *deny*.

Default

The default for *submitwithformaterrors* is **deny**.

validoverlap

This feature adjusts the tolerance of the overlap test that the Viewer performs when signing items. The overlap test detects overlaps between signed items and unsigned items. Because the unsigned items might be moved later to reveal information that changes the meaning of the form, the overlap test prevents the user from signing these forms. However, in some cases the overlap is very slight, and should be allowed.

By default, the Viewer allows a 2 pixel overlap between items. While you can use **validoverlap** to change this value, you should explore the following solutions first:

- The overlap test looks for signed items that overlap unsigned items. You should always follow the best practice of signing as much of the form as possible. This means that if your form is failing the overlap test, you should first consider whether you can sign additional items to correct this problem.
- Lines can often cause problems. In general, you should either sign all lines in the form, or sign none of the lines in the form. This will help you avoid problems with lines that overlap each other. If you choose to sign none of the lines in the form, you may need to adjust the placement of some lines to prevent them from overlapping with other items, such as fields. While this may seem like a lot of work, it's preferable to signing the overlapping line. Signing the line may create more problems than it solves, as the line may overlap with a number of other lines, which may overlap with further lines, and so on. Because of the work involved in determining which lines need to be signed, it's best to avoid this scenario if possible.

If you still find that you cannot sign your form because of overlapping items, you should adjust the **validoverlap** setting by small increments (such as 1 or 2 pixels) until you can successfully sign the form or until you reach an overlap of 8 pixels.

Note: Never set the **validoverlap** to allow more than 8 pixels of overlap. If you find that you still cannot sign your form with an overlap of 8 pixels, you should redesign your form so that the items do not overlap as much. Using a larger **validoverlap** increases the risk that a malicious user could successfully modify a signed form.

The validoverlap can also be set as low as 0 pixels. The default of 2 pixels accounts for the size of a border, which is included in the size of each item even if the border is not drawn.

Example

```
<ufv_settings>
   <validoverlap>4</validoverlap>
</ufv_settings>
```

A form with this setting would allow signed items to overlap unsigned items by 4 pixels or less.

Default

The default setting for *validoverlap* is **2** pixels.

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

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

Program Number:

Printed in USA

S325-2610-00