IBM® Workplace Forms™

Version 2.6.1

Embedding the Viewer in HTML Web Pages

S325-2598-00

Note

Before using this information and the product it supports, read the information in "Notices," on page 15.

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

Embedding the Viewer in HTML Web Pages

Embedding an XFDL form in an HTML page allows users to view forms inside portal environments or as part of a series of web pages. Furthermore, it allows application designers to manipulate the embedded form like any other HTML object. Unlike other embedded objects, such as HTML or XML forms, embedded XFDL forms maintain user data even after the web page is changed or refreshed.

This document explains how to embed an XFDL form inside an HTML page using the HTML *object* element and associated *parameter* attributes. It also describes how to refer to the *object* parameters from inside the form and recommends best practices for enclosing signed forms.

System Requirements

To view XFDL forms embedded in HTML pages, users must have Workplace Forms™ Viewer version 2.5 or higher or a PureEdge-branded Viewer 6.2 or higher installed on their computer. Additionally, the web pages can be viewed with Internet Explorer 6, Netscape 7.x, Mozilla 1.x, or Firefox 1.x.

Overview

Embedding a form in an HTML page allows you to display forms in a portal environment or as part of a series of web pages. This permits forms to be updated dynamically in response to user selections on the HTML page.

To embed a form, you must use two HTML elements:

- **objects** — An object allows you to display non-HTML data in the browser. It also allows you to define the size and borders of an object.
- **scripts** — The script contains and loads the form.You can also use other scripts to contain data that is used to modify the form, such as XML instances.

The HTML *object* is a placeholder. In your HTML file, it specifies the location where the object will be displayed in relation to the other elements on the web page. For example, you could place an object inside an HTML table cell:

```
<tr>
   <td>
      <object>...</object>
   </td>
</tr>
```

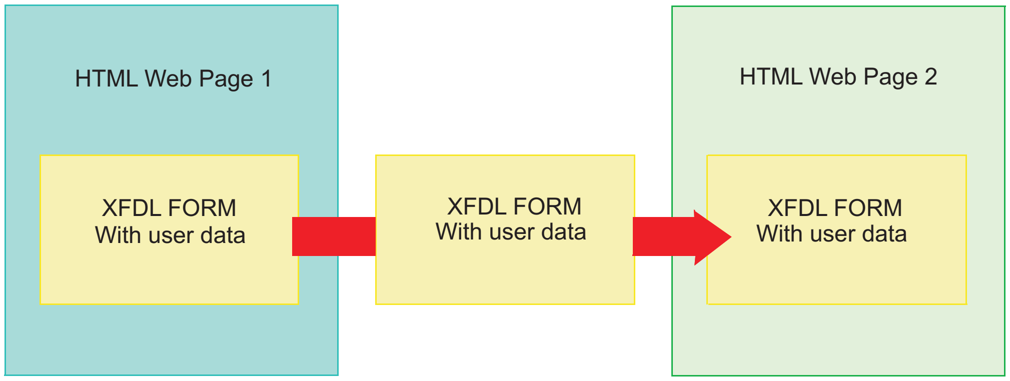
The *script* element is not displayed as part of the web page, so it can be placed anywhere in your HTML code. You may prefer to place scripts directly after the *object* that they are associated with, or at the beginning or end of the page. In the following example, the script that contains the form is placed immediately after the table with the form's object:

```
</table>
   <script>
      ... XFDL form ...
   </script>
```

Displaying Forms in an HTML Page

When you embed an XFDL form in an HTML page, you are essentially declaring that a portion of the web page is controlled by a program other than the browser, such as Workplace Forms Viewer. As a result, this portion of the page operates differently than the HTML page in which it resides.

For example, if a user is filling out a form on one webpage but needs more information to complete it, the user may click a link on the HTML page which takes them to a new page. Normally, if users performed this action, any data they entered into the form would be lost. However, with an XFDL form, the form can simply be *detached* from the first web page and provided for viewing in a second web page, as shown in the diagram below:



When a form is detached from one web page and reattached to another, it maintains all of the user's data. Unlike the first web page, any successive web pages that need to display the same form do not need to explicitly embed the form. If you want them to display the same form and retain user data, they must simply contain objects that share the same *detach_id*.

Successive pages do not have to display the same form. A second web page can display an entirely different form if it contains a different object. You may even redirect users to an entirely different web page if users are taking too long to complete a form and the detached form has timed out. Objects that contain a *refresh_URL* ensure that users can be directed to another web page if they need help or to a fresh new form if they abandoned completion of the old one.

Once the form is embedded in an HTML web page, you can treat it like any other HTML object. For example, you can use javascript, other portlets, or servlets to generate dynamic content within the object or form. For example, you can generate new XML instances for your form.

Embedding a Form

To embed a form inside an HTML page, you must:

- Determine which browsers you want to support.
- Create an object inside the web page.
- Define the object parameters.
- Create and define a nested object (if necessary).
- Define the script parameters of the embedded form.
- Insert the XFDL form you want to embed.
- Add XML instances to the web page. (optional)

Selecting Which Browsers to Support

You can embed a form into any HTML page. However, every browser interprets HTML code differently. If you know that all your users are only using one type of browser, you can focus your HTML code to support it specifically. On the other hand, if your users are using multiple browser types, you will need to create additional code to ensure that all of the browsers recognize and display the embedded form.

If you are only supporting one kind of browser, you can create a single object. If you are supporting multiple browser types, you need to create nested objects. In this case, the outer object must always support Internet Explorer, while the inner object must support the Mozilla, Netscape, and Firefox browsers.

Creating an Object

The *object* element in HTML allows web page designers to specify data to be rendered by a browser plugin. IBM® has taken advantage of this element to allow XFDL forms to be embedded in a web page. Using the *object* element allows you to specify how the object is implemented and the location of the object's data. This is done using the following attributes:

Attribute	Browser Support	Description
id	All	Assigns a name to the object, identifying it for manipulation by associated applications. It must be unique within the HTML page. For example, `<OBJECT id="unique_name">`
classid	IE only	Specifies the browser plugin used to display the object. In this case, it identifies the Active X control that activates the Viewer. This value must always be: `CLSID:354913B2-7190-49C0-944B-1507C9125367` For example: `<OBJECT  classid="CLSID:354913B2-7190-49C0 -944B-1507C9125367">` Note that the **classid** attribute essentially fullfills the same function for the Internet Explorer browser as the **type** attribute does for the Netscape, Firefox, and Mozilla browsers.
type	Mozilla, Netscape, Firefox	The *type* attribute has one parameter: • **mime type** – Identifies the language used by the script for Internet Explorer 6.x. This value must always be **application/vnd.xfdl**. For example: `<OBJECT type="application/vnd.xfdl">` Note that the **type** attribute essentially fullfills the same function for the Netscape, Firefox, and Mozilla browsers as the **classid** attribute does for the Internet Explorer browser.
height	All	Indicates the height of the object in pixels. For example: `<OBJECT height="400">`
width	All	Indicates the width of the object in pixels. For example: `<OBJECT width="800">`

Attribute	Browser Support	Description
style	All	Allows you to specify font color, styles, and sizes, as well as background colors and object positioning. For example: `<OBJECT STYLE="position:absolute; left:10px;top:10px;height:600px; width:800px;background-color:aqua; font-family:Helvetica">`

You may also modify the form's display area with the following optional attributes:

Attribute	Browser Support	Description
align	All	Sets the position of the object within its allowed display area. The valid settings are: *left*, *right*, and *center*. For example: `<OBJECT align="center">`
border	All	Sets the width of the object's border in pixels. This value must always be an integer. For example: `<OBJECT border="2">`
hspace	All	Sets the amount of white space (in pixels) to be inserted to the left and right of an object. If the object has a border, this additional white space is outside of the border. This number is always an integer. For example: `<OBJECT hspace="2">`
vspace	All	Sets the amount of white space (in pixels) to be inserted above and below an object. If the object has a border, this additional white space is outside of the border. This number is always an integer. For example: `<OBJECT hspace="2">`

Note: Remember, if you plan to support multiple browsers, the outer object must support Internet Explorer.

Example

In the following example, the object is assigned a unique name, pointed at the Viewer Active X control, and given a size.

```
<OBJECT id="unique_name" height="400" width="800"
   classid="CLSID:354913B2-7190-49C0-944B-1507C9125367">
   ... object parameters...
</OBJECT>
```

Note: For more information about the HTML object element, refer to the W3C website at: http://www.w3.org/TR/REC-html40/struct/objects.html.

Defining the Object Parameters

Object attributes identify and set the size of the object, but to specify run-time values you must use the *param* element. The *param* element consists of *name* and *value* pairs. For example:

```
<PARAM NAME="XFDLID" VALUE="XFDLData">
```

These *param* attributes have no meaning in HTML. Instead, their *properties* determine the Viewer's behavior and how it handles the embedded form. There are six of these special properties:

XFDLID
The ID of the *script* element that contains the form. This value can be anything, as long as the *XFDLID* and the *script id* match. For example:

```
<PARAM NAME="XFDLID" VALUE="XFDLData">
```

detach_id
The unique ID of the form instance. This attribute allows an XFDL form to 'detach' from one web page and 'attach' itself to another while retaining any entered user data. The *detach_id* value can be any unique string. For example:

```
<PARAM NAME="detach_id" VALUE="1234567890ABCD">
```

Successive objects containing the same form must all have the same *detach_id* if you want to maintain user data between web pages. For example, if form A appears in HTML pages 1- 4, and the form object in page 1 has a detach_id of 10236B, then the form objects in pages 2 - 4 must also have a *detach_id* of 10236B.

If you specify a *detach_id*, you must also give the object a *TTL* (Time To Live).

refresh_URL
The URL called to refresh the HTML page if the object's *detach_id* has expired and the page does not have an embedded XFDL form. This URL can point to any web page, regardless of whether it contains an object or a form. *refresh_URL* does not maintain user data. For example:

```
<PARAM NAME="refresh_URL"
   VALUE="http://www.serv1/IRS/Sched22.htm">
```

TTL
The length of time the detached form will live before being destroyed automatically (called the Time To Live). The default *TTL* is 0. This value is given in seconds. For example:

```
<PARAM NAME="TTL" VALUE="15">
```

If you specify a *TTL*, you must also give the object a *detach_id*.

retain_viewer
Determines whether the Viewer remains available after completing Viewer *replace* or *done* actions. If *retain_viewer* is **off**, the Viewer closes after completing either action. If it is **on**, the Viewer remains available for further use, such as displaying a new form or to retaining form data after a submission. For example:

```
<PARAM NAME="retain_Viewer" VALUE="on">
```

instance_1... instance_n
Identifies XML instances inside an HTML page. This instances can be used to modify specified XML instances inside the XFDL form. This information includes:

- The ID of the new instance.
- The ID of the form instance.

Three additional values may be included:

- *xforms* – Indicates that the instance data is XForms.

- *replace* – Indicates that the new instance data replaces the original instance data. This value is optional, but either *replace* or *append* must be used.
- *append* – Indicates that the new instance data is added to the original instance data. This value is optional, but either *replace* or *append* must be used.
- An XPath reference – Indicates where the new data should be placed. This could be a non-default instance, or a particular element in an instance. Note that any namespaces listed in this value resolve relative to the document root. This value is optional if only one data instance exists. If multiple instances exist, this reference is mandatory.

For example:

```
<PARAM NAME="instance_1"
   VALUE="new_Inst old_Inst xforms; append '[custom:rec][custom:name]'">
```

The contents of the *value* attribute must be space separated.

Multiple instance parameters must have sequentially numbered names, starting with 1. For example, *instance_1*, *instance_2*, and so on.

Example

The following example displays an object with a complete set of *param* attributes to control the Viewer's behavior:

```
<OBJECT...object attributes...>
   <PARAM NAME="XFDLID" VALUE="theForm">
   <PARAM NAME="TTL" VALUE="15">
   <PARAM NAME="detach_id" VALUE="12354678901234567890">
   <PARAM NAME="refresh_URL" VALUE="http://www.serv1/IRS/Sched22.htm">
   <PARAM NAME="retain_Viewer" VALUE="on">
   <PARAM NAME="instance_1" VALUE="newIns oldIns xforms; replace [0][0]">
</OBJECT>
```

Note: The *param* element requires a start tag only. It does **not** require an end tag. For more information about the HTML parameter element, refer to the W3C website at: http://www.w3.org/TR/REC-html40/struct/objects.html#h-13.3.2.

Order of Precedence

Objects frequently include multiple parameters. As a result, the Viewer must follow an order of precedence so that the object parameters are always processed in a consistent manner. When a new web page containing a form object is opened, the following precedence is used to determine what is displayed:

1. **detach_id** — If a *detach_id* exists and its *TTL* has not expired, the object displays the form referenced by the *detach_id*.
2. **XFDLID** — If there is no *detach_id* or it has expired, the object displays the embedded form identified by the *XFDLID*.
3. **refresh_URL** — If there is no *detach_id* or *XFDLID*, then the entire page is reloaded to display the page specified by the *refresh_URL*.

Adding a Nested Object

If you intend to support multiple browsers, you need to create and define a second object inside the first. This nested object must support the Netscape, Mozilla, and Firefox browsers.

To create a nested object, you need to add a second object inside the start and end tags of the first object. For example:

```
<OBJECT id="ObjectForIE" height="200" width="200" border="5"
   classid="CLSID:354913B2-7190-49C0-944B-1507C9125367">
   <parameters>

   <OBJECT ID="ObjectForFF" height="200" width="200" border="5"
      type="application/vnd.xfdl">
      <parameters>
   </OBJECT>

</OBJECT>
```

You must also add an IF statement that around the second object that identifies it as being for non-IE browsers. This statement must be contained inside a comment wrapper. For example:

```
<!--[if !IE]>-->
   <object2>
<!--<![endif]-->
```

Example

The following example displays the nested objects required to support both Internet Explorer and Mozilla-based browsers:

```
<OBJECT id="ObjectForIE" height="200" width="200" border="5"
   classid="CLSID:354913B2-7190-49C0-944B-1507C9125367">
   <PARAM NAME="XFDLID" VALUE="XFDLData">
   <PARAM NAME="detach_id" VALUE="2507088000">
   <PARAM NAME="refresh_url" VALUE="envAware.html">
   <PARAM NAME="TTL" VALUE="17">
   <PARAM NAME="retain_Viewer" VALUE="on">

<!--[if !IE]>-->

   <OBJECT ID="ObjectForFF" height="200" width="200" border="5"
      type="application/vnd.xfdl">
      <PARAM NAME="XFDLID" VALUE="XFDLData">
      <PARAM NAME="detach_id" VALUE="2507088000">
      <PARAM NAME="refresh_url" VALUE="envAware.html">
      <PARAM NAME="TTL" VALUE="17">
      <PARAM NAME="retain_Viewer" VALUE="on">
   </OBJECT>

<!--<![endif]-->

</OBJECT>
```

Defining the Script Parameters

In HTML, the *script* element places a script inside the HTML page. In the case of XFDL forms, the script includes the form itself. There are a number of attributes that modify the *script* element. These attributes help the Viewer to recognize the embedded form as valid XFDL. They include:

language
Identifies the language used by the script for older versions of Internet Explorer. This value must always be **XFDL**. For example:

```
<SCRIPT language="XFDL">
```

id Identifies the object referred to by the script. This id *must* match the object's *XFDLID* value. For example, if your *XFDLID* property reads:

```
<PARAM NAME="XFDLID" VALUE="theForm">
```

then your *script id* must be:

```
<SCRIPT id="XFDLData">
```

type The *type* attribute has five parameters:

- **mime type —** Identifies the language used by the script for Internet Explorer 6.x. This value must always be **application/vnd.xfdl**.
- **wrapped** — Identifies the type of container that encloses the form. This value must always be **comment**.
- **next-chunk** — Indicates the next portion of the form to Mozilla, Firefox, or Netscape browsers. If your form is larger than 64 K, these browsers require your form to be broken into 64 K 'chunks' to ensure consistent display. This value must be the *script id* of the next portion of the form.
- **encoding** — Indicates that the contents of the script are compressed. This parameter is mandatory if the Viewer is to be embedded in Mozilla, Firefox, or Netscape browsers; optional parameter for Internet Explorer browsers. In either case, its value is always **base64**.
- **content-encoding** — Indicates internationalized characters. If this optional parameter is used, its value identifies the type of Unicode encoding that is in use. For example, UTF-16.

For example:

```
<SCRIPT type="application/vnd.xfdl; wrapped=comment;
   encoding=base64; next-chunk=Part2; content-encoding=utf-16">
```

These parameters must be separated with a semi-colon followed by a space.

Example

The following example displays a *script* starting tag, complete with all of the required attributes:

```
<SCRIPT language="XFDL" id="XFDLData"
   type="application/vnd.xfdl; wrapped=comment; encoding=base64;
   content-encoding=utf-16">
      ...enclosed form...
</SCRIPT>
```

Note: For more information about the HTML script element, see http://www.w3.org/TR/REC-html40/interact/scripts.html.

Adding the Form to the Script

Once you've defined the script parameters, you can insert your XFDL form.

To insert the form:

1. Place a comment wrapper between the *script* tags. For example:

```
<SCRIPT ...script attributes...>
   <!--
   -->
</SCRIPT>
```

2. Paste the entire XFDL form inside the comment wrapper.

Example

The following example depicts a small XFDL form enclosed in a comment wrapper:

```
<SCRIPT ...script attributes...>
   <!--
      <?xml version="1.0"?>
      <XFDL xmlns="http://www.ibm.com/xmlns/prod/XFDL/7.0"
         xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom">
         <globalpage sid="global">
             <global sid="global">
                <vfd_date>9/7/2004</vfd_date>
            </global>
         </globalpage>
         <page sid="PAGE1">
            <global sid="global">
               <label>PAGE1</label>
            </global>
            <label sid="LABEL1">
               <value>This is a short sample form.</value>
            </label>
         </page>
      </XFDL>
   -->
</SCRIPT>
```

Adding XML Instances to the HTML Page

You can include XML instances (XForms or XML Data Model) in your web page that contain XML data you can move into your form. This XML data can replace or add to existing XML data inside your form – particularly useful for dynamically populating forms with information selected by the user from an HTML page.

For example, consider a purchase order form embedded in an HTML page. The XFDL form is a generic purchase order form, possibly personalized with user information via a data fragment. This form is displayed in an HTML page (or a series of HTML pages) that allow users to select the products they want to purchase. When users select the desired product, they trigger the application server to create a custom XML instance to replace the generic XML instance currently in the XFDL form. To the eyes of the users, the form appears to automatically populate with the correct purchase information, including order number, product type, and pricing.

Regardless of whether the XML instance script in hardcoded into the HTML page or generated by an application server, it must be wrapped in a *script* container. Like a *script* element containing an XFDL form, you must define an instance's *script* with the following attributes:

language
Identifies the language used by the script for older versions of Internet Explorer. This value must always be **XFDL**. For example:

```
<SCRIPT language="XFDL">
```

id Identifies the instance. This id *must* match the object's *instance* value. For example, if *instance_1* reads:

```
<PARAM NAME="instance_1"
   VALUE="new_Instance old_Instance
   append[custom:record][custom:name]">
```

then your *script id* must be:

```
<SCRIPT id="new_Instance">
```

type The *type* attribute has two parameters:

- **mime type —** Identifies the language used by the script for Internet Explorer 6.x. This value must always be **application/vnd.xfdl**.
- **wrapped** — Identifies the type of container that encloses the instance. This value must always be **comment**.
- **encoding —** Indicates that the instance is compressed. This parameter is mandatory for all instances. The value is always **base64**.

For example:

```
<SCRIPT type="application/vnd.xfdl;wrapped=comment">
```

The type and wrapped parameters must be separated with a semi-colon.

Important: To identify XForms instances, you must add an *xforms* parameter to the form object. For more information, see "Defining the Object Parameters" on page 4.

To embed an instance:

1. Create a new *script* element. For example:
```
<SCRIPT id=new_Instance type="application/vnd.xfdl; wrapped=comment">
</SCRIPT>
```
2. Place a comment wrapper between the script tags.
3. Paste the XML instance between the comment tags.

Example

The following example depicts a *script* element that contains an XML instance enclosed in a comment wrapper:

```
<SCRIPT id="new_Instance" type="application/vnd.xfdl; wrapped=comment">
   <!--
      <name xmlns="http://www.ibm.com/xmlns/prod/XFDL/Custom">
         Arnold Jevaston</name>
   -->
</SCRIPT>
```

Order of Precedence

Forms often include multiple XML instances. They may even refer to *data fragments*, frequently used scraps of XML data that are stored on users' computers. If your HTML contains replacement instances, or if replacement instances are generated by an application server, the Viewer must follow an order of precedence to ensure the instances are always loaded in a consistent manner. When a new web page containing XML instances is opened, the following precedence is used to determine what is displayed:

- **Data Fragments** — If Smartfill is on and an appropriate data fragment is stored on a user's computer, data fragments are loaded first. If there is a replacement instance targeted at the fields populated by the Smartfill data, the data contained in the replacement instance is ignored. To avoid this sort of complication, ensure that your data fragments and instances have unique names or different naming styles for each. For example, if your organization uses a data fragment called *PersonalInfo*, but you want to use a different instance containing personal data in a form, make sure the form's instance has a different name or naming convention. For example, *personal_data*.
- **instance_1...instance_n** — If your form is embedded in an HTML page that contains multiple XML instances, the Viewer will process them sequentially, in accordance to the number sequence in their *script id*. Thus *instance_1* will load first, *instance_2* will load second, and so on.

Sample Embedded Form

The following example shows all the code required to fully embed a form. This includes an *object* element with all of its mandatory attributes, the object parameters, the *script* element, and a sample XFDL form wrapped in a comment structure.

```
<OBJECT id="Object1" height="200" width="200" border="5"
   classid="CLSID:354913B2-7190-49C0-944B-1507C9125367">
   <PARAM NAME="XFDLID" VALUE="XFDLData">
   <PARAM NAME="detach_id" VALUE="2507088000">
   <PARAM NAME="refresh_url" VALUE="envAware.html">
   <PARAM NAME="TTL" VALUE="17">
   <PARAM NAME="retain_Viewer" VALUE="on">
</OBJECT>
<SCRIPT language="XFDL" id="XFDLData"
   type="application/vnd.xfdl; wrapped=comment">
   <!--
      <?xml version="1.0"?>
         <XFDL xmlns="http://www.ibm.com/xmlns/prod/XFDL/7.0"
            xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom">
            <globalpage sid="global">
               <global sid="global">
                  <vfd_date>9/7/2004</vfd_date>
               </global>
            </globalpage>
            <page sid="PAGE1">
               <global sid="global">
                  <label>PAGE1</label>
               </global>
               <label sid="LABEL1">
                  <value>This is a short sample form.</value>
               </label>
            </page>
         </XFDL>
    -->
</SCRIPT>
```

Sample Embedded XForms Form

The following example shows all the code required to fully embed an XForms form. This includes nested *object* elements that support both IE and Mozilla-based browsers.

```
<OBJECT id="forIEbrowser" height="300" width="800" border="5"
   classid="CLSID:354913B2-7190-49C0-944B-1507C9125367">
   <PARAM NAME="XFDLID" VALUE="theForm">
   <PARAM NAME="detach_id" VALUE="2507088000">
   <PARAM NAME="refresh_url" VALUE="envAware.html">
   <PARAM NAME="TTL" VALUE="17">
   <PARAM NAME="retain_Viewer" VALUE="on">
   <PARAM NAME="instance_1" VALUE="newInst xforms; replace=&quot;.&quot;">

<!--[if !IE]>Mozilla 1.x, Firefox 1.x, Netscape 7+ and others will use the
   nested inner object-->

<OBJECT ID="forMozillabrowser" height="300" width="800" border="5"
   type="application/vnd.xfdl">
   <PARAM NAME="XFDLID" VALUE="theForm">
   <PARAM NAME="detach_id" VALUE="2507088000">
   <PARAM NAME="refresh_url" VALUE="envAware.html">
   <PARAM NAME="TTL" VALUE="17">
   <PARAM NAME="retain_Viewer" VALUE="on">
   <PARAM NAME="instance_1" VALUE="newInst xforms; replace=&quot;.&quot;">
</OBJECT>
```

```
<!--<![endif]-->

</OBJECT>

<SCRIPT language="XFDL" id="theForm2" type="application/vnd.xfdl; wrapped=comment;
   next-chunk=chunk2">
   <!--
   <?xml version="1.0" encoding="UTF-8"?>
      <XFDL xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom"
         xmlns:ev="http://www.w3.org/2001/xml-events"
         xmlns:xfdl="xmlns:xfdl="http://www.ibm.com/xmlns/prod/XFDL/7.0"
         xmlns:xforms="http://www.w3.org/2002/xforms"
         xmlns="xmlns:xfdl="http://www.ibm.com/xmlns/prod/XFDL/7.0">
         <globalpage sid="global">
            <global sid="global">
               <xformsmodels>
                  <xforms:model>
                     <xforms:instance id="instance1" xmlns="">
                        <root>
                           <field_1></field_1>
                           <field_2></field_2>
                           <field_3></field_3>
                        </root>
                     </xforms:instance>
                  </xforms:model>
               </xformsmodels>
            </global>
         </globalpage>
      -->
</SCRIPT>

<SCRIPT language="XFDL" id="chunk2" type="application/vnd.xfdl; wrapped=comment;
   next-chunk=chunk3">
      <!--
         <page sid="PAGE1">
            <global sid="global">
               <label>PAGE1</label>
            </global>
            <field sid="field1">
               <xforms:input ref="field_1">
                  <xforms:label>Field with xforms:input</xforms:label>
               </xforms:input>
            </field>
            <field sid="field2">
               <xforms:textarea ref="field_2">
                  <xforms:label>Field with xforms:textarea</xforms:label>
               </xforms:textarea>
              <size>
                 <width>30</width>
                 <height>2</height>
              </size>
           </field>
        -->
</SCRIPT>

<SCRIPT language="XFDL" id="chunk3" type="application/vnd.xfdl; wrapped=comment">
        <!--
           <field sid="field3">
              <xforms:secret ref="field_3">
                  <xforms:label>Field with xforms:secret</xforms:label>
              </xforms:secret>
           </field>
           <spacer sid="vfd_spacer">
              <itemlocation>
                 <x>250</x>
                 <y>250</y>
                 <width>1</width>
```

```
                    <height>1</height>
                 </itemlocation>
              </spacer>
           </page>
        </XFDL>
     -->
     </SCRIPT>
```

Using Object Parameters in Computes

The Viewer functions enable you to trigger actions in the Viewer. The *param* Viewer function allows you to call specific object parameters in computes that return the parameter's value. For example, if the *param* function called the object's *XFDLID* parameter, then it would return the object's XFDL ID. For more information, see the guide entitled "*Viewer Functions*".

Embedding Signed Forms

If you want to embed a signed form into a web page, the form must be compressed in base 64. When Internet Explorer parses a signed embedded form, it may add extra carriage returns to the form. This prevents the form from matching the signature hash value, which causes the Viewer to declare that the form's signature is invalid. Compressing signed forms ensures that this does not occur.

The easiest way to compress an XFDL form is to open it and compress it in the Designer.

To compress a form:

1. Open Workplace Forms Designer.
2. Open the form you want to compress.
3. In the **Outline** view, select **Form Global**.
   - **Form Global** data appears in the **Properties** view.
4. Right-click the **Menu** ▽ icon, and select **Show Advanced Properties**.
5. Open the **Advanced** tree, and browse to **saveformat**.
6. Click the popup in the field next to **saveformat** and select **application/vnd.xfdl;content-encoding=″base64-gzip″.**

Embedding Internationalized Forms

If your form contains characters outside of the ASCII (0-7F) range, the character encoding in your XFDL form must match the BSTR encoding passed by the browser to the object containing the Viewer. The easiest way to ensure that they match is to specify character encoding only in the HTML form. You can place it in a *meta* tag inside the HTML *head* tag. For example:

```
<META http-equiv="Content-Type" content="text/html;charset=utf-8">
```

Furthermore, you must also remove the character encoding already in your form. As the Designer automatically adds character encoding, you must open the form in another text editor, such as UltraEdit, Notepad, or TextPad, and delete the encoding.

To strip character encoding from an XFDL form:

1. Open the file in a text editor. Do not open it in Workplace Forms Designer.

2. Search for the *encoding* attribute on the *xml version* tag and delete the attribute and its setting.
3. Save the form.

However, under some circumstances you may find that you must include character encoding in your XFDL form. If so, you must still ensure that the form encoding matches the BSTR encoding. Therefore, if you include character encoding in your form, it must always be:

- UTF-16LE

You cannot change your form's character encoding in the Designer. You must edit it in another text editor, such as UltraEdit or Notepad.

To change your form's character encoding:

1. Open the file in a text editor.
   - Do not open it in the Designer.
2. Search for the encoding attribute on the xml version tag.
3. Change the encoding setting to UTF-16LE.
4. Save the form.

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.

Trademarks

Program Number:

Printed in USA

S325-2598-00