IBM® Workplace Forms™

Version 2.6.1

Duplicating Form Nodes

S325-2606-00

Note
Before using this information and the product it supports, read the information in "Notices," on page 11.

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

Duplicating Form Nodes

This document provides a full description of the three functions used in duplicating form nodes within a form. These functions are: *duplicate*, *generateUniqueName*, and *getReference*. These functions are contained in the standard *system* function package.

The following sections will introduce you to the concept of duplicating form nodes and show you how these three functions can be used together. For a quick reference to each of these functions, refer to the last pages of this document.

You should be familiar with XFDL and the XFDL compute system (including function calls) before you use these functions. Refer to the *XFDL Specification* for more information. Also, you should have some understanding of how the components of an XFDL form are organized into a hierarchy of interrelated nodes. Refer to the *Workplace Forms™ Server — API User's Manual* for more information about the node structure.

Why Duplicate Form Nodes?

The *duplicate* function allows form users to generate new pages or items within a form open in Workplace Forms Viewer. As a form developer, you might use the *duplicate* function to allow users or applications that use XFDL forms to dynamically create new items and pages as needed on a form.

The node duplication process works as follows:

1. A custom option must be set up to call the *duplicate* function under specified circumstances (for example, when an action is initiated or a button is clicked). The custom option can appear anywhere.
2. The *duplicate* function creates a new node with a unique sid. The parameters specified in the *duplicate* function determine the identity of the template node, the location of the duplicate, and the sid of the duplicate.
3. Once the duplication is complete, the Viewer automatically refreshes its display to include any visible changes. If a new page was added, the Viewer will continue to display the original page, and the user will need to use a paging control to move to the new page.

What is a Form Node?

A form node refers to a uniquely identifiable portion of a form, such as the form itself, a page, an item, an option or an array element. You can use the functions in this document to duplicate pages, items and options.

For those unfamiliar with the concept of nodes, here is a brief description of the terminology used in this document. You may wish to view the diagram on the next page as you read these terms.

Original Node
The node you want to duplicate.

New Node
The node you create by duplicating the original node.

Parent Node

The node that the original node belongs to. For example, the *parent* node of an item would be the page on which the item is located, and the parent node of a page would be the form.

Child Node

This refers to any pages, items, options or array elements contained within the original node. For example, an item node may contain several *children*, which would be the options describing that item. Each option, in turn, may itself have several children the form of array elements.

Sibling Node

Refers to any node that is a child of the same parent as the original node's parent. For example, if a page contains three items, INS_LABEL, NAME_FIELD and SAVE_BUTTON, then NAME_FIELD and SAVE_BUTTON are *sibling* nodes of INS_LABEL.

Partial Reference

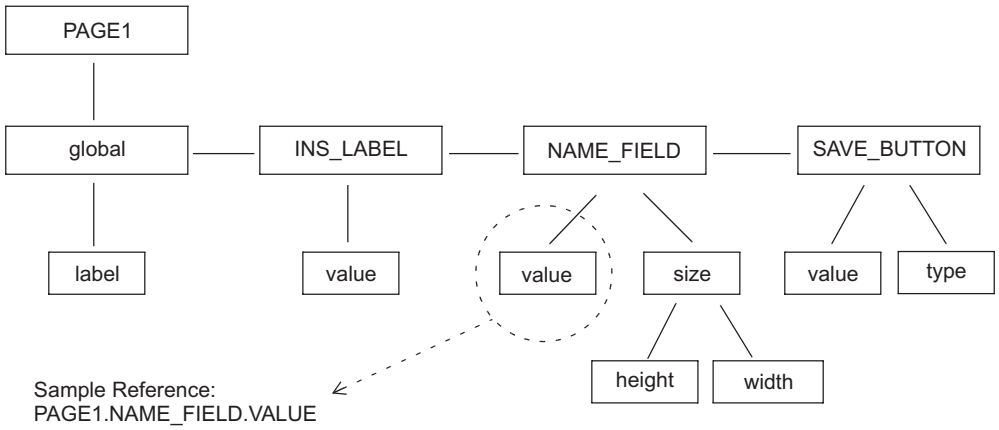
This term refers to an incomplete page, item or option reference. For example, a partial reference to PAGE1.LABEL2.value would be simply **value**.

The following code and diagram illustrate how the node structure works with XFDL forms. In this example, both the sample code and the node structure represent the same form:

Sample Code

```
<XFDL xmlns="http://www.ibm.com/xmlns/prod/XFDL/7.0"
   xmlns:xfdl="http://www.ibm.com/xmlns/prod/XFDL/7.0">
   <globalpage sid="global">
      <global sid="global">
      </global>
   </globalpage>
   <page sid="PAGE1">
      <global sid="global">
      </global>
      <label>PAGE1</label>
      <label sid="INS_LABEL">
         <value>Please enter your name:</value>
      </label>
      <field sid="NAME_FIELD">
         <value></value>
         <size>
            <width>30</width>
            <height>1</height>
         </size>
      </field>
      <button sid="SAVE_BUTTON">
         <value>Save Form Now</value>
         <type>saveform</type>
      </button>
   </page>
</xfdl>
```

Node Structure



About Unique Identifiers

Every node within a form requires a unique identifier (for example, PAGE1 or SAVE_BUTTON). This is referred to as a scope identifier or sid. Each sid must be unique within its parent node.

Potential problems arise when nodes are duplicated. If a node were to be duplicated more than once, each copy would possess the same sid as the first copy. Using the function *generateUniqueName* in concert with the *duplicate* function eliminates this problem.

When duplicating pages, you must also take into account the fact that new pages will contain the identical options and items as their originals. This is particularly significant for the *label* option at the page level, which determines the text shown in the title bar of Workplace Forms Viewer. When a page is duplicated, the label is also duplicated. Thus the new page node contains the same label as the original, and the identity of the new page is not apparent to the user viewing the form, who sees only the page label. To get around this, the *label* option of the page should be set to display the page sid. However, since the pages this effects do not exist when you write the code, you need some way of dynamically referencing the page sid.

You can use the function *getReference* to determine the page sid. *getReference* will return the reference to a page, item, or option node.

In the following example, the value of the *label* option in PAGE1 will be hard-coded as ″PAGE1″. If this page was duplicated, the new page might have a sid of New_PAGE1. The label for the page, and the title bar in the Viewer, however, would still display ″PAGE1″.

```
<page sid="PAGE1">
   <global sid="global"></global>
   <label>PAGE1</label>
```

By using *getReference* to dynamically update the label option, this can be avoided. In the example below, the label will display the sid of the page.

```
<page sid="PAGE1">
   <global sid="global"></global>
   <label compute="getReference('label', 'option', 'page')"></label>
```

The *getReference* function can be used any time a node reference would need to be updated after a duplication.

Creating Paging Controls for Duplicated Pages and Setting the Tab Order for Duplicated Items

If your form creates new pages, users will not be able to get to those pages unless paging controls are created as well. However, since the pages do not exist when you are creating the controls, you need some way of referencing duplicated pages. Likewise, it is difficult to set the tab order for items that do not exist yet.

To accomplish these tasks, you need to use the relative references provided by XFDL. These references let you refer to items or pages which either do not yet exist, or vary depending on how the form is configured. The available relative references are:

Relative Tag	Refers to	Reference Syntax
pagefirst	First page in form description.	global.pagefirst->item
pagelast	Last page in form description.	global.pagelast->item
pageprevious	Previous page in form description. First page points to the last page in the form.	global.pageprevious->item
pagenext	Next page in form description. Last page points to first page in the form.	global.pagenext->item
itemfirst	First item in page description.	global.itemfirst->option
itemlast	Last item in page description.	global.itemlast->option
itemprevious	Previous item in page description. First item points to last item in page description.	global.itemprevious->option
itemnext	Next item in page description. Last item points to first item in page description.	global.itemnext->option

Example

If you are creating a paging control on a page that may be duplicated, you must use a dynamic reference rather than a hard-coded URL option to move to the next page.

```
<button sid="PAGING_CONTROL">
   <type>done</type>
   <url compute="global.pagenext"></url>
   <value>Next Page</value>
</button>
```

When this button is duplicated, the URL is still valid because it points to the next page, no matter what that page is. If the URL is on the last page of the form, it will cycle back to the first page, and vice versa.

Note: You do not need to use the complete dereference syntax for a page change as you would if you wanted to reference a specific item on the next page.

For more information on this topic, see the section on Relative References in the *XFDL Specification*.

Duplicating XForms Nodes

If you duplicate an entire page, the elements in the new page will automatically bind to the XForms data model. However, this binding will not occur if you only duplicate an item or option within the page. This means that duplicating items or options that are bound to the data model may cause your forms to behave erratically.

getReference

getReference returns the reference to a page, item, or option. This function works as a "Where am I?" check for these form elements. It is always called from within the node for which the reference is needed, but the returned reference can be for any parent node down to that from which it was called. For instance, *getReference* could be called from a custom option within a field, but return only the reference for that field.

Call

```
getReference('theReference', 'referenceStart', 'referenceLevel')
```

Parameters

Expression	Type	Description
theReference	String	A string that contains a partial reference to the current node that you wish to identify. This node is either the node from which the function is called, or a sibling node of the node from which the function is called. If you want the reference for the node from which the function was called, use that node. For instance, to find the reference for a *value* option in a field, **theReference** would be **value**.
referenceStart	page | item | option	A string that contains the type of node identified in theReference. For instance, if **theReference** is value, **referenceStart** will be option.
referenceLevel	page | item | option	A string that contains the lowest node level in the reference for the function to return. If **getReference** is called from an option node, but **referenceLevel** is identified as item, the reference returned would be **page.item**.

Returns

A reference to the page, item, or option from which the function was called.

Example

In the following example, *getReference* returns a reference to the node at the option level. The result is PAGE1.field1.value.

```
<page sid="PAGE1">
   <global sid="global"></global>
   <field sid = "field1">
      <value compute="getReference('', '', 'option')"></value>
   </field>
</page>
```

In the following example, *getReference* returns a reference to the node at the page level. The result is: PAGE1.

```
<page sid="PAGE1">
   <global sid="global"></global>
   <field sid = "field1">
      <value compute="getReference('', '', 'page')"></value>
   </field>
</page>
```

generateUniqueName

generateUniqueName will create a unique name for a page, item or option. Use this function to generate a unique name for any node created within a form.

Call

```
generateUniqueName(theReference, referenceStart, namePrefix)
```

Parameters

Expression	Type	Description
theReference	String	A string that contains a reference to the parent node of the node that you wish to generate a unique name for. For instance, if you want to generate a unique name for an item on PAGE_1, **theReference** would be **PAGE_1**. To indicate that the parent node is the current form, leave this parameter blank. You would do this if you wanted to generate a unique name for a page.
referenceStart	page | item | option	A string that contains the type of node identified in **theReference**. In the example above, **referenceStart** would be page.
namePrefix	String	A string that describes the prefix for the unique name generated. This can be any string you like. If you choose new_item as the **namePrefix**, the first name generated by the function would be **new_item1**, the second would be **new_item2**, and so on.

Returns

A unique page, item or option name consisting of a *namePrefix* plus an integer.

Notes

- *generateUniqueName* generates a name consisting of the *namePrefix* and an integer. The function then searches the parent node specified by *theReference* and *referenceStart* for any page, item, or option identifiers with the same name. If the name already exists in the parent node then the function increments the integer until a unique name can be generated.

Example

The custom option *generate* calls the *generateUniqueName* function to generate a unique name and sets the result in new_Name.

```
<button sid="duplicateFieldButton">
   <value>duplicate Field1</value>
   <new_Name></new_Name>
   <custom:generate xfdl:compute="toggle(activated, 'off', &#xA;
      'on')== '1' ? set('new_Name', generateUniqueName &#xA;
      ('PAGE1', "page", newField_")) : ''"></custom:generate>
 </button>
```

duplicate

This function makes a copy of a specified form node. The *duplicate* node returned from this function can be attached to any other node as either a sibling or a child. You can also assign a new identifier to the new node, as indicated by the **theIdentifier** parameter. All of the properties of the original node are duplicated, including any children. For instance, a duplicated page will contain all the settings, items and options of the original.

Call

```
duplicate('theReference', 'referenceStart', 'baseNode',
   'baseNodeStart', 'where', 'theIdentifier')
```

Parameters

Expression	Type	Description
theReference	String	A string that contains the reference to the node you wish to duplicate. For instance, if you want to duplicate FIELD3 on PAGE2, **theReference** would be **PAGE2.FIELD3**.
referenceStart	form | page | item | option	A string that contains the type node identified in the **theReference**. In the example above, **referenceStart** would be **item**.
baseNode	String	A string that contains the identity of the reference node for the new node. In the example above, if you wanted the new node to also be on PAGE2, the **baseNode** would be **PAGE2**. If you wanted the new node to be added to the page as a sibling to FIELD8, the **baseNode** would be **FIELD8**.
baseNodeStart	form | page | item | option	A string that contains the node type identified in baseNode. In the first example above, **baseNodeStart** would be **page**; in the second it would be **item**.

where	String	A description of the desired location of the new node in relation to the supplied **baseNode**. Valid settings are: **APPEND_CHILD** — Adds the new node as the last child of the 'baseNode'. You may also use APPEND CHILD without the underscore between the words. **AFTER_SIBLING** — Adds the new node as a sibling of the 'baseNode', placing it immediately after that node in the form structure. You may also use AFTER SIBLING without the underscore between the words. **BEFORE_SIBLING** — Adds the new node as a sibling of the 'baseNode', placing it immediately before that node in the form structure. You may also use BEFORE SIBLING without the underscore between the words.
theIdentifier	String or form item reference	An identifier for the new node. If this parameter is left out, the same identifier that was used in **theReference** is used. You can call *generateUniqueName* within duplicate to provide this identifier.

Returns

The duplicate node.

Notes

- The *duplicate* function uses the API function *UFLDuplicate* to generate copies of form nodes within a form.
- The node that the *duplicate* function copies is specified by two parameters **theReference** and **referenceStart**.
- The node that you wish to attach the copy to is specified by two parameters **baseNode** and **baseNodeStart**.
- Normally, the function *generateUniqueName* is used to generate a new name for each node that is duplicated. This function may be called within **theIdentifier** parameter.
- If you are duplicating options or array elements, they do not need new unique identifiers, but they must be placed in a different item or option node.
- If you duplicate an entire page, the elements in the new page will automatically bind to the XForms data model. However, this binding will not occur if you only duplicate an item or option within the page. This means that duplicating items or options that are bound to the data model may cause your forms to behave erratically.

Example

This example is broken up into several steps to make the duplication process as clear as possible.

An item is created whose value is simply its item reference. The item reference is defined by calling the function *getReference*. For more details about *getReference* refer to the function description provided in this document.

```
<page sid="PAGE1">
   <global sid="global"></global>
   <field sid="Field1">
      <label>myField</label>
      <value compute="getReference('value','option', 'option')"></value>
   </field>
```

Create a button that duplicates ″Field1″ and places it on ″PAGE2″.

```
<button sid="duplicateFieldButton">
      <value>duplicate Field1</value>
      <type>select</type>
```

Within the button, the custom option *generate* calls the *generateUniqueName* function to generate a unique name and sets the result in *new_Name*. It is possible to omit this step and use *generateUniqueName* directly in **theIdentifier** parameter in the *duplicate* function, but here the steps are separated for clarity. For more details about this function refer to the function description provided in this document.

```
   <custom:new_Name></custom:new_Name>
      <custom:generate xfdl:compute="toggle(activated, 'off', 'on') &#xA;
         == '1' ? set('new_Name', generateUniqueName('PAGE1',  &#xA;
         'page', 'newField_')) : '' "></custom:generate>
```

The custom option duplicate calls the *duplicate* function to make a copy of ″Field1″ and assigns the new field the name specified in new_Name. The new field is placed after the last item on PAGE2.

```
   <custom:duplicate xfdl:compute="toggle(activated, 'off','on') == '1' ? &#xA;
      duplicate('PAGE1.Field1', 'item', 'PAGE2', 'page', &#xA;
      'append_child', custom:new_Name) : '' "></custom:duplicate>
</button>
```

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

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Program Number:

Printed in USA

S325-2606-00