IBM® Workplace Forms™

IBM

Version 2.6.1

Creating Signature Buttons in XFDL

S325-2604-00

Note
Before using this information and the product it supports, read the information in “Notices,” on page 25.

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.

© 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

Creating Signature Buttons in XFDL . . 1
What the User Sees 1
Signature Basics 2
Signing a Form 3
Setting the Text that the Button Displays 4
Making Signatures Mandatory 4
Signing Portions of Forms 5
About the Signature Filters 6
Order of Precedence of Signature Filters 7
Elements to Exclude from Signatures 8
Using Filters to Guarantee Form Security 9
Setting the Signature Type 9
Using Generic RSA Signatures 10
Using Microsoft CryptoAPI Signatures 10
Using Netscape Signatures 11
Using Clickwrap Signatures 11
Using Authenticated Clickwrap Signatures . . . 14
Using Signature Pad Signatures. 15
Using Entrust Signatures 17
Using Silanis Signatures 18
Reference: Available Options for Digital Signature Buttons 23

Appendix. Notices 25
Trademarks 26

© Copyright IBM Corp. 2003, 2006

Creating Signature Buttons in XFDL

This document explains how to create signature buttons in Extensible Forms Description Language (XFDL). It illustrates the various types of signing ceremonies, describes how to build a basic signature button, and introduces signature filters for increasing the flexibility of signatures and the security of signed forms.

This document is intended for forms designers and assumes that you have a basic knowledge of XFDL.

It contains the following sections:

- ″**What the User Sees**″ — Explains the signing ceremony from the user's perspective, and demonstrates how a signature prevents further modification of the form.
- ″**Signature Basics**″ — Introduces the elements of a signature and provides sample code for a basic signature button.
- ″**Setting the Text that the Button Displays**″ — Explains how to set the button to display different text depending on whether the form has been signed.
- ″**Making Signatures Mandatory**″ — Explains when and why you should make signatures mandatory, as well as providing sample code and usage details.
- ″**Signing Portions of Forms**″ — Introduces the concept of signature filters and how they can enhance security. It provides descriptions of the different types of filters and how you can use them, as well as supplying sample code.
- ″**Setting the Signature Type**″ — Provides an overview of signature types, details available engines, and presents sample code.
- ″**Reference: Available Options for Digital Signature Buttons**″ — Provides a list of all the options you can use with signature buttons.

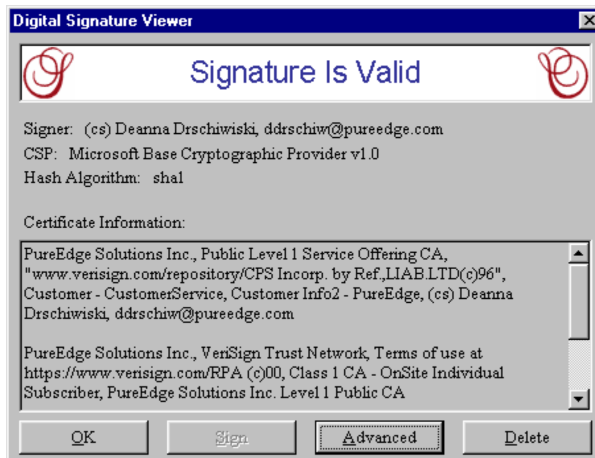
What the User Sees

To sign a form, users click the signature button. Typically, signature buttons are recognized by the text they display, or a label near the button. They also tend to be larger than the text, since they also have to display the signer's name once they are signed.

When users click the signature button, the Digital Signature Viewer opens. The Digital Signature Viewer allows users to sign forms, verify or delete existing signatures, and view the details of what parts of the form were signed.

© Copyright IBM Corp. 2003, 2006

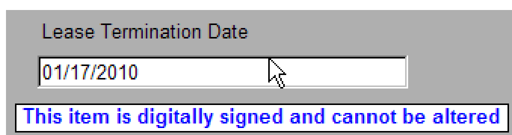
Users click **Sign** to sign the form. If they have more than one certificate installed on their computer, a dialog box appears, displaying all of the available signatures and allowing them to choose one. Once users have selected a certificate, the form is signed and the Digital Signature Viewer displays the details of the signature.



User's then click **OK** to return to the form. In the form, the signature button changes to reflect the new signature. Typically, it displays the name and email address of the signer (although this depends on the type of signature engine used).

(cs) Deanna Drschiwiski, ddrschiw@pureedge.com

Once a form is signed, the Viewer prevents users from changing any of the signed information. Furthermore, when the user mouses over a signed item in a form, a tooltip appears indicating that the item has been signed and cannot be altered.



Signature Basics

A typical signature button is created using the following options:

```
<button sid="BUTTON1">
   <type>
   <value>
   <format>
```

```
      <size>
      <signformat>
      <signature>
   </button>
```

These options define the characteristics of the signature button in the following ways:

- **type —** Setting the *type* option to *signature* indicates that this is a signature button.
- **value —** The *value* option sets the text that the button displays. In general, you will display an instruction, such as "Click here to sign", or the signer's identity (typically the signer's name and email address).
- **format —** Use *format* to make the signature mandatory, so that the user must sign the button before submitting the form.
- **size —** Use *size* to set the size of the button. We recommend that you set signature buttons to be at least 40 characters wide. This ensures that there is room to display the signer's name and email address on the button.
- **signformat —** Use *signformat* to set which type of signature you want the button to create, such as Netscape, Microsoft®, ClickWrap, and so on.
- **signature —** The *signature* option sets the name of the *signature* item that will be created when the form is signed.

In addition to the options listed above, signature buttons also use a number of *filters* that determine which portions of the form are signed. These filters are created by adding a combination of filtering options to the button. These filtering options are discussed in greater detail later in this document.

Signing a Form

When a user clicks a signature button, the Viewer automatically creates the following elements in the form:

- A *signer* option is added to the signature button. In general, this option contains the signer's name and email address. However, the value assigned to this option depends on the signature engine that was used to sign the form.
- A *signature* item is created on the same page as the signature button. This item is given the name indicated in the button's *signature* option, and is used to store the details of the signature.

While you do not need to create the *signature* item yourself, the following code shows the options typically used in a *signature* item:

```
   <signature sid="SIGNATURE1">
      <layoutinfo>
      <signformat>
      <signature>
      <fullname>
      <mimedata>
   </signature>
```

The *signature* item will also include any of the filtering options used in the signature button.

Setting the Text that the Button Displays

Signature buttons switch between two states: unsigned and signed. When a signature button is unsigned, it is generally good practice to have the button display instructions, such as "Click here to sign". When a signature button is signed, it is generally a good practice to show the identity of the person who signed it.

Since buttons display the text in their *value* option, you must set the *value* option appropriately, depending on whether the button is unsigned or signed. To do this, you must use a compute.

The compute for this relies on the button's *signer* option. As discussed earlier, the *signer* option is created by the Viewer when the button is signed. This means that there is no *signer* option when the button is unsigned.

With this in mind, you can create a simple test to set the value of the button. If the *signer* option has no value (that is, if it does not exist), then there is no signature and the button should read "Click here to sign". If the signer option has a value, then there is a signature and the button should show the identity of the signer. Since the *signer* option stores the signer's identity, the button's *value* can bet set to equal the *signer* option.

More formally, you might express this logic as: if the *signer* option is empty, then set the button's *value* to "Click here to sign"; otherwise, set the button's *value* to equal the *signer* option.

In XFDL, you would write this as follows:

```
<button sid="BUTTON1">
   <value compute="signer == '' ? 'Click to sign' : signer"></value>
</button>
```

Making Signatures Mandatory

Most signatures are required. That is, most documents that offer a place to sign also *require* a signature before they can be accepted or processed.

Similarly, you may want to make your signature buttons mandatory. When you make the signature button mandatory, the Viewer will prevent the user from submitting the form until they have signed it. This ensures that you do not receive unsigned forms.

You can use the *format* option to make signatures mandatory. In this case, the *format* option should contains two elements:

- **datatype —** Allows you to set the data type to *string*. This means that the *value* of the button (the text that is displayed by the button) will be a string. For example:

  ```
  <datatype>string</datatype>
  ```

- **constraints —** Allows you to place constraints on items, such as templates, lengths, and ranges. You can also use *constraints* to make the button mandatory. The format of *constraints* is:

  ```
  <constraints>
     <settingname1>setting</settingname1>
        ...
     <settingnamen>setting</settingnamen>
  </constraints>
  ```

In this case, you only want to ensure that the signature button is mandatory, so that the user must sign the button before they can submit the form. Therefore, you must add the *mandatory* element. The format of *mandatory* is:

```
<mandatory>on|off</mandatory>
```

The *mandatory* element must be contained inside the *constraints* element, as shown:

```
<constraints>
   <mandatory>on</mandatory>
</constraints>
```

For example, the format option would be written as:

```
<button sid="BUTTON1">
   <format>
      <datatype>string</datatype>
      <constraints>
         <mandatory>on</mandatory>
      </constraints>
   </format>
</button>
```

Once you have set the button to be mandatory, you may also need to change the *value* option of the button. This is because the mandatory feature is triggered off of the *value* option of the item. If the item has no value, the Viewer will force the user to provide a *value*. If the item has a *value*, the Viewer will allow the user to continue.

Earlier, we discussed setting unsigned buttons to read "Click here to sign". However, if we assign a *value* to a button that is unsigned, the mandatory feature will not work correctly. Thus, we must modify the logic that sets the button's value option.

In this case, we want to use the following logic: if the *signer* option is empty, then set the button's *value* to empty; otherwise, set the button's *value* to equal the *signer* option.

For example, a button with the correct *format* and *value* options would be written as:

```
<button sid="BUTTON1">
   <format>
      <datatype>string</datatype>
      <constraints>
         <mandatory>on</mandatory>
      </constraints>
   </format>
   <value compute="signer == '' ? '' : signer"></value>
</button>
```

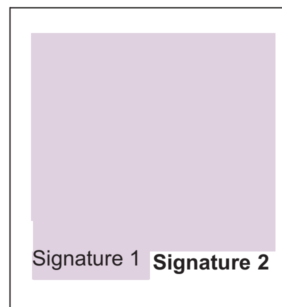
Note: This works for forms that have a single signature. If your form requires multiple signatures, there are more sophisticated methods that you must use to make each signature become mandatory in turn.

Signing Portions of Forms

Forms are frequently signed by more than one person. For example, many forms include a "For Office Use Only" section that requires a second signature by one of the staff processing the form. In these cases, the office worker must be able to enter more information in the unsigned portion of the form and then add their own signature.

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

Signature 1 signs the filled section of the form.



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

XFDL makes this possible by using signature filters. Signature filters specify which parts of the form a particular signature will sign. This means that you can create one signature that sign only the first portion of the form, and then a second signature that signs the second portion of the form, or the entire form.

Note: If you do not use any signature filters, the entire form is signed by default.

About the Signature Filters

When creating filters, you must first decide what type of filtering you want to use:

- **keep** — Keep filters let you specify the portions of the form you want to sign, leaving the rest of the form unsigned.
- **omit** — Omit filters let you specify the portions of the form you don't want signed, and ensures that the rest of the form is signed.

  Note: Omit filters provides greater security than keep filters, and should be used as a best practice.

Once you have decided what sort of filtering to use, you can create your filters by combining the various filtering options. The filtering options are:

- ***signitems* option** — Specifies the types of items that the signature will keep or omit. For example, you might set the filter to omit all *button* items from the signature.
- ***signoptions* option** — Specifies the types of options that the signature will keep or omit. For example, you might set the filter to omit all *triggeritem* options from the signature.
- ***signgroups* option** — Specifies one or more groups, as defined by the *group* option, that the signature will either keep or omit. This filters any radio buttons or cells belonging to that group, but does not filter *list*, *popup*, or *combobox* items. For example, if you had a popup containing a cell for each State, you might set the filter to omit the *State* group, which would omit all cells in that group.
- ***signdatagroups* option** — Specifies one or more datagroups, as defined by the *datagroup* option, that the signature will either keep or omit. This filters data items belonging to that datagroup, but does not filter any *action*, *button*, or *cell* items. For example, if you had an enclosure button containing references, you might set a filter to omit the *References* datagroup, which would omit all data items in that group.

- ***signinstance* option —** Specifies the XForms instance data that the signature will either keep or omit. For example, you might set the filter to omit any data that is not sent to the server.
- ***signitemrefs* option —** Specifies individual items that the signature will keep or omit. For example, you might set the filter to omit BUTTON1 on PAGE1.
- ***signoptionrefs* option —** Specifies individual options that the signature will keep or omit. For example, you might set the filter to omit BUTTON1.value on PAGE1.
- ***signnamespaces* option —** Specifies all of the form elements and attributes in the indicated namespace that the signature will keep or omit. For example, you might set the filter to omit the http://www.ibm.com/xmlns/prod/XFDL/Custom namespace.
- ***signpagerefs* option —** Specifies individual pages that the signature will keep or omit. For example, you might set the filter to omit PAGE1.

Each filtering option must be set to either keep or omit, and must include a list of elements. For example, to omit all buttons and labels from a signature, you would use the following filter:

```
<button sid="BUTTON1">
   <signitems>
      <filter>omit</filter>
      <itemtype>button</itemtype>
      <itemtype>label</itemtype>
   </signitems>
</button>
```

Signatures can include any number filters. Each filter acts in an order of precedence. For example, the *signitems* filter is always processed first. The *signoptions* filter is then only applied to those items that remain.

For example, to omit all *button* and *label* items, as well as omitting all *bgcolor* options in the remaining items, you would use the following filter:

```
<button sid="BUTTON1">
   <signitems>
      <filter>omit</filter>
      <itemtype>button</itemtype>
      <itemtype>label</itemtype>
   </signitems>
   <signoptions>
      <filter>omit</filter>
      <optiontype>bgcolor</optiontype>
   </signitems>
</button>
```

For detailed description of the order of precedence, see "Order of Precedence of Signature Filters".

Order of Precedence of Signature Filters

Complex forms frequently use a combination of filter options. As a result, the Viewer must follow an order of precedence so that filter options are always processed in a consistent manner. The following table lists the behavior of the filter options and the order in which the Viewer applies them:

Filter Option	Omit Filter	Keep Filter	Notes
1. *signinstance*	Omits only data in the indicated instance; throws them out.	Keeps only data in the indicated instance; throws others out.	
2. *signnamespaces*	Omits only elements and attributes in the namespaces indicated; throws them out.	Keeps only elements and attributes in the namespaces indicated; throws others out.	An element is kept if any of its children are kept, even if it is in the wrong namespace.
3. *signitems*	Omits only the types listed; omitted items are not signed.	Keeps only the types listed; all other types are not signed.	
4. *signoptions*	In the items that remain (not omitted by *signitems*) omits all listed options. Omitted options are not signed.	In the items that remain, keeps all indicated options. All other options remain unsigned.	
5. *signpagerefs*	Omits the specified pages. Overrides settings in *signitems* and *signoptions.*	Keeps the specified pages. Respects settings in *signitems* and *signoptions.*	Omitted pages are not completely deleted; the page sid is preserved.
6. *signdatagroups* and *signgroups*	Omits the items in that group, even if they are of a type that should be kept according to a *signitems* setting.	Keeps the items in that group, even if they are of a type that should not be kept according to a *signitems* setting.	These settings override those in *signpagerefs*.
7. *signitemrefs*	Omits the specified items and overrides previous settings.	Keeps the specified items and overrides previous settings. Respects settings in *signoptions*.	These settings override *signitems*, *signgroups*, *signpagerefs*, and *signdatagroups*.
8. *signoptionrefs*	Omits the specified options, overriding any previous settings.	Keeps the specified options, overriding any previous settings. If the item containing the option has been omitted, that item's sid and the specified option are preserved.	This option's setting overrides *all* other filter options.

When using signatures, note that the *mimedata* option is always omitted in the following scenarios, regardless of the signature filters in use:

- The *mimedata* option in a *signature* item is always omitted from the signature that item represents.
- The *mimedata* option in a *data* item that stores a signature image (see the *signatureimage* option) is always omitted from the signature that image represents.

Elements to Exclude from Signatures

There are certain form elements that you should always exclude from signatures. For example, when you click a submit button, the *triggeritem* option is

automatically set, recording the name of the button that triggered the submission. However, if a signature been applied to the *triggeritem* option, the Viewer will not be able to update the option correctly.

In general, you should always exclude the following form elements from a signature:
- the *triggeritem* option
- the *coordinates* option
- any portion of the form that subsequent users will change
- subsequent signatures and signature buttons
- the signature item that stores the information for the signature you are creating

Using Filters to Guarantee Form Security

Digital signatures protect the legal validity of signed forms. If you do not use best practices when creating your signature filters, you increase the chance that users will be able to repudiate signed forms based on a misunderstanding about what they've signed.

For example, a mortgage contract with a flawed signature filter could allow tampering to terms that are crucial to the contract. This alteration could confuse or modify the terms of the contract, and result in a challenge to the validity of the document.

Judicious use of omit and keep filters can guarantee form security. As a best practice, you should always use the *omit* filter to create signatures. If you use a keep filter, it should only be to sign another signature that is already using an omit filter.

Using these practices will help prevent you from accidentally excluding items and options that should be signed, and will prevent malicious users from adding to the form's contents without breaking the signature.

Setting the Signature Type

XFDL supports a number of different signature engines. Each signature button must be configured to use a specific engine when signing the form. The available engines are:
- Generic RSA
- Authenticated Clickwrap
- Microsoft CryptoAPI
- Signature Pad
- Netscape
- Entrust
- Clickwrap
- Silanis

You can specify the signature engine for your form by using the *engine* parameter in the *signformat* option. If you do not include a *signformat* option or do not set the *engine* parameter, the form will use the Generic RSA engine by default.

The syntax for using each engine is described in the following sections.

Using Generic RSA Signatures

The Generic RSA engine is the default signing engine, and is used if you do not declare an engine in the *signformat* option. It automatically searches for any standard RSA certificates on the user's computer, which includes both Microsoft CryptoAPI and Netscape certificates.

To create a signature button that uses the Generic RSA engine, set the following parameters in the *signformat* option:

- **MIME type —** The MIME type that is used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine —** The name of the signing engine to use. In this case, **Generic RSA**.
- **delete —** Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

For example, the XFDL code for a button that uses the Generic RSA engine and prevents the signature from being deleted looks like this:

```
<button sid="BUTTON1">
   <signformat>
      application/vnd.xfdl;
      engine="Generic RSA";
      delete="off"
   </signformat>
</button>
```

Using Microsoft CryptoAPI Signatures

The CryptoAPI engine uses certificates located in the Microsoft certificate store. To create a signature button that uses the CryptoAPI engine, set the following parameters in the *signformat* option:

- **MIME type —** The MIME type that is used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine —** The name of the signing engine to use. In this case, *CryptoAPI*.
- **delete —** Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

CryptoAPI also uses optional parameters to set specific CSP behavior. You should only set these parameters if you need to use a specific CSP. In all other cases, you should accept the default values. The parameters are:

- **csp —** The cryptographic service provider used to create the signature. This should only be set if you need to use a specific CSP. Otherwise, the signature will default to the Microsoft Base Cryptographic Service Provider.
- **csptype —** Identifies the type of CSP in use. This should only be set if you need to use a specific CSP. Otherwise, the signature will default to a full RSA implementation (*rsa_full*).

For example, the generic XFDL code for a button using the CryptoAPI engine looks like this:

```
<button sid="BUTTON1">
   <signformat>
      application/vnd.xfdl;
      engine="CryptoAPI"
   </signformat>
</button>
```

Using Netscape Signatures

The Netscape engine uses certificates located in the Netscape certificate store. Although Netscape browsers are not supported by the Viewer, Firefox browsers use a Netscape certificate store. To create a signature button that uses the Netscape engine, set the following parameters in the *signformat* option:

- **MIME type —** The MIME type that will be used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine —** The name of the signing engine to use. In this case, **Netscape**.
- **delete —** Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

Netscape also uses an optional parameter to set the hash algorithm used to generate the signature. This parameter is:

- **hashalg —** The name of the hash algorithm to use when generating the signature. Valid algorithms are *sha1* and *md5*. The default algorithm is *sha1*.

For example, the generic XFDL code for a button using the Netscape engine looks like this:

```
<button sid="BUTTON1">
   <signformat>
      application/vnd.xfdl;
      engine="Netscape"
   </signformat>
</button>
```

Using Clickwrap Signatures

Clickwrap signatures are electronic signatures that do not require digital certificates. While they still offer a measure of security due to an encryption algorithm, Clickwrap signatures are not security tools. Instead, Clickwrap signatures offer a simple method of obtaining electronic evidence of user acceptance to an electronic agreement. The Clickwrap signing ceremony authenticates users through a series of questions and answers, and records the signer's consent. Clickwrap style agreements are frequently found in licensing agreements and other online transactions.

To create a signature button that uses the Clickwrap engine, set the following parameters in the *signformat* option:

- **MIME type —** The MIME type that is used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine —** The name of the signing engine to use. In this case, **ClickWrap**.
- **delete —** Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

For example, the XFDL code for a generic signature button using the Clickwrap engine looks like this:

```
<button sid="BUTTON1">
   <signformat>
      application/vnd.xfdl;
      engine="ClickWrap"
   </signformat>
</button>
```

This creates a minimal Clickwrap signing ceremony with default options, as shown:

You can add more features to the signing ceremony by using additional parameters. For example, you may want to ask the signer some questions to help confirm their identity, and then have them type some text to confirm their agreement. These additional parameters are discussed in the next section.

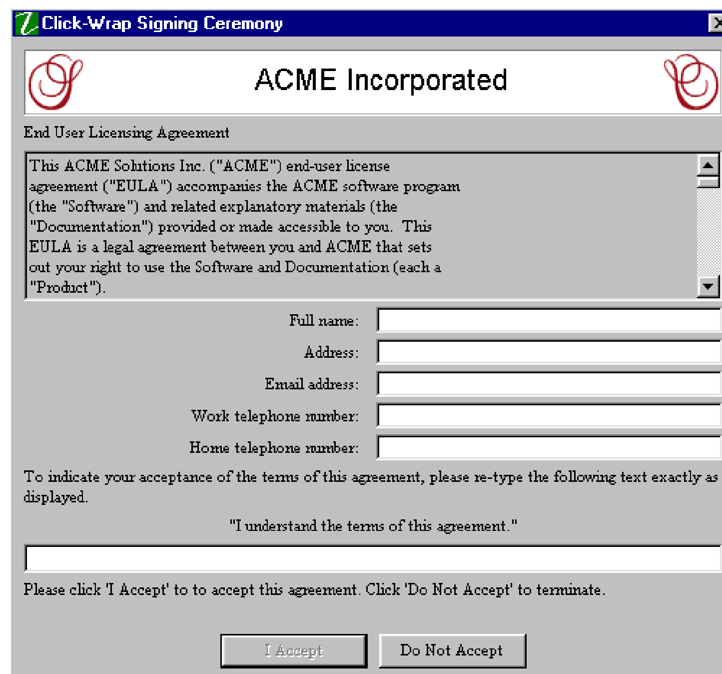
Customizing Your Clickwrap Signing Ceremony

You can customize your Clickwrap signing ceremony by adding parameters to the *signformat* option. The following table lists the optional parameters, their use, and their default settings:

Parameter	Use	Default Text
titleText	Sets the title of the signing ceremony. This is generally used to describe the signing ceremony, the company, or the title of the agreement.	Click-Wrap Signing Ceremony
mainPrompt	Typically used to explain the signing ceremony to users.	None
mainText	Normally contains the main text of the agreement. For example, the text of a licensing agreement. You can add as much text as necessary to this parameter; the signing ceremony automatically displays scrollbars if the text is longer than the display field.	None
question1Text - question5Text	Allows you to ask from one to five questions that help establish the identity of the user.	None
answer1Text - answer5Text	These are the answers to the questions asked by question1Text to question5Text. The Viewer normally adds these parameters to the form as users complete the fields. However, you can prepopulate the answers if you want.	None
echoPrompt	Use this to instruct the user to echo the echoText. Generally, if you include echoText, you will want this to say something like, "Please type the following phrase to show that you understand and agree to this contract."	None
echoText	This is the actual text that the user should echo, or re-type. Generally, this should say something like, "I understand the terms of this agreement."	None

Parameter	Use	Default Text
buttonPrompt	This is an instruction line that appears above the accept and reject buttons. The user must click the accept button to sign, so generally the prompt should say something like, "Click accept to sign this document."	Click the Accept button to sign.
acceptText	Sets the text that the accept button displays.	Accept
rejectText	Sets the text that the reject button display.	Not Accept

The following diagram illustrates a customized Clickwrap signing ceremony that uses all of the optional parameters:



The following is the XFDL code that created the above signing ceremony. Note that the values of some parameters, such as the *mainText*, have been shortened from the example above.

```
<button sid="BUTTON1">
   <signformat>
      application/vnd.xfdl;
      engine="ClickWrap";
      titleText="ACME Incorporated";
      mainPrompt="End User License Agreement";
      mainText="This ACME Solutions...";
      question1Text="Full name:";
      question2Text="Address:";
      question3Text="Email address:";
      question4Text="Work telephone number:";
      question5Text="Home telephone number:";
```

```
      echoPrompt="To indicate your acceptance...";
      echoText="I understand the terms of this agreement."
   </signformat>
</button>
```

Using Authenticated Clickwrap Signatures

Authenticated Clickwrap enables users to securely sign a form without relying on an extended PKI infrastructure. In normal use, the user signs the form by entering an ID and secret, such as a password. When the form is sent to the server, the server retrieves the user's secret from a database and uses that secret to verify the signature. Furthermore, the server can notarize the Authenticated Clickwrap by signing it with a digital certificate, thereby creating a secondary digital signature. This secondary signature shows that the server has confirmed the identity of the signer, and ensures that the original signature can be trusted over time.

Note: Authenticated Clickwrap is a separately licensed product. Please ensure that your company has the license to use Authenticated Clickwrap before you provide forms or functionality that rely on it.

Authenticated Clickwrap signatures use all of the parameters that Clickwrap signatures support. However, in practice you will probably only use the *question1Text* through *question5Text*, and some additional parameters that are unique to Authenticated Clickwrap.

To create a signature button that uses the Authenticated Clickwrap engine, set the following parameters in the *signformat* option:

- **MIME type —** The MIME type that is used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine —** The name of the signing engine to use. In this case, **HMAC-ClickWrap**.
- **delete —** Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

In addition to these parameters and the parameters used by Clickwrap signature, Authenticated Clickwrap signatures also use the following parameters:

- **HMACSigner —** Indicates which answers identify the signer. The answer is always written as $answer_n$. For example, if *question1Text* asked the user's name, then *answer1* would identify the signer. Note that this parameter is mandatory.
- **HMACSecret —** Indicates which answers contain the secret. The answer is always written as $answer_n$. For example, if *question2Text* asked for the user's secret, then *answer2* would identify the signer. Note that this parameter is mandatory.
- **readonly —** Indicates which answers are read-only. This is useful if you have prepopulated the form, and want to ensure certain answers cannot be changed. The answer is always written as $answer_n$. For example, if you wanted to make the first answer read-only, then you would use *answer1*.

Note: The HMACSigner and the HMACSecret cannot point to the same answer. Furthermore, if you list more than one answer, you must separate the answers with a comma. Be sure not to add any additional white space, such as a space. For example, ″answer1,answer2″ is correct.

For example, the following code shows a signature button that will request the user's ID and password:

```
<button sid="BUTTON1">
   <signformat>
      application/vnd.xfdl;
      engine="HMAC-ClickWrap";
      question1Text="Enter your ID:";
      question2Text="Enter your password:";
      HMACSigner="answer1";
      HMACSecret="answer2";
   </signformat>
</button>
```

Using Signature Pad Signatures

These signatures use a signature pad that plugs into your computer. The signature pad allows the user to create a hand-written signature that is then applied to the form. Before you can use these signatures, you must have:

- The Signature Pad Extension for the Workplace Forms™ Viewer
- A signature pad
- Software that enables the signature pad

The Signature Pad Extension is available as a separate install package that adds support for signature pads to the Viewer.

Signature pads and their supporting software are available from a variety of vendors. Workplace Forms Viewer supports the signature pads from the following companies:

- Interlink
- Topaz
- Any WinTab compliant signature pad

Once you have all of these components, you can create and use Signature Pad signatures.

Creating a Signature

To create a signature button that uses a signature pad, set the following parameters in the *signformat* option:

- **MIME type —** The MIME type that is used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine —** The name of the signing engine to use. This is **SignaturePad**.
- **delete —** Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

Signature Pad signatures use all of the parameters that Authenticated Clickwrap signatures support, with the exception of HMACSigner and HMACSecret. As well, Signature Pad signatures use the following additional parameters:

- **tsp —** Forces a particular Signature Pad signing engine. Valid setting are: Interlink, Topaz, and WinTab. If you do not set this parameter, the Viewer will determine which signature pads are available, and will always use either Interlink or Topaz in preference to WinTab.
- **startText —** Sets the text that is displayed by the button that starts the capture of the hand-written signature. The default text is ″Start Capture″.
- **endText —** Sets the text that is displayed by the button that ends the capture of the hand-written signature. The default text is ″End Capture″.

- **penColor —** Sets the color of the pen that is used to write the signature. This parameter will accept either a color name, such as ″blue″, or a comma-separated RGB value, such as ″11, 12, 13″. The default value is ″0, 0, 0″, which is black.
- **Background Color —** Sets the background for the signature. This parameter will accept either a color name, such as ″blue″, or a comma-separated RGB value, such as ″11, 12, 13″. The default value is ″192, 192, 192″, which is a shade of grey.

Displaying the Signature

Unlike other signature types, Signature Pad signatures create an image of the handwritten signature that you must display in the signature button. This means that you must set up the button so that it can properly display the image. Additionally, your form must include two other images, one that you can display when the signature is blank and one that you can display when the signature is invalid.

When you design the form, include both the blank and invalid image in data items in the form. This will make them available to the signature button.

Sizing the Signature Button and Scaling the Image

In most cases, the signer's handwritten signature will be larger than the standard button size for a digital signature. To account for this, you should:

- Make the signature button larger than normal.
- Use the *imagemode* option to scale the image to the correct size.

To make the signature button larger, use the size option to change it's height. You may want to experiment to determine the appropriate size.

To ensure that the image fits within the button, you should also set the *imagemode* option to **scale**. This will resize the image (either by shrinking it or enlarging it) to a best fit within the button, and will maintain the original aspect ratio.

For example, the following code creates a button that is 40 characters long and 5 lines tall, and that scales the image to fit the button:

```
<button sid="BUTTON1">
   <size>
      <width>40</width>
      <height>5</height>
   </size>
   <imagemode>scale</imagemode>
</button>
```

Setting the Name of the Signature Image

When the signature is created, the Viewer will automatically create a data item that stores the signature's image. You use the *signatureimage* option to set the name of the data item and create a link to it from the button. When the Viewer creates the data item for the image, it will automatically give the item the name that you set in this option.

For example, to create a data item called SIGIMAGE, you would use the following code:

```
<button sid="BUTTON1">
   <signatureimage>SIGIMAGE</signatureimage>
</button>
```

Displaying the Signature Image

Once you have created the signature button, you must link it to the signature images in your form and ensure that the correct image is displayed. To do this, you add an *image* option to the button. This will link the button to one of the three images: blank, invalid, or the signature itself.

To make the decision, you must add a compute to the image option. This compute will rely on the *signer* option in the button: if the *signer* option is empty, then the form has not been signed and the *image* should be set to the blank image; if the *signer* option is "INVALID", then the signature is invalid and the *image* should be set to the invalid image; and finally, if the *signer* option is anything else (which means it is the signer's name), then the image should be set to the signature itself.

The following example shows a compute that makes this decision. In this case, the data items that contain the images are named BLANKIMAGE, INVALIDIMAGE, and SIGIMAGE:

```
<button sid="BUTTON1">
   <image compute="signer == '' ? 'BLANKIMAGE' : &#xA;
      (signer == 'INVALID' ? 'INVALIDIMAGE' : &#xA;
      'SIGIMAGE')"></image>
</button>
```

Putting it All Together

The following example shows a signature button that is completely configured for a Signature Pad signature. In addition, the code includes the BLANKIMAGE data item and an INVALIDIMAGE data item.

```
<button sid="BUTTON1">
   <type>signature</type>
   <signformat>
      application/vnd.xfdl; engine="SignaturePad"
   </signformat>
   <size>
      <width>40</width>
      <height>5</height>
   </size>
   <imagemode>scale</imagemode>
   <signatureimage>SIGIMAGE</signatureimage>
   <image compute="signer == '' ? 'BLANKIMAGE' : &#xA;
      (signer == 'INVALID' ? 'INVALIDIMAGE' : &#xA;
      'SIGIMAGE')"></image>
</button>
<data sid="BLANKIMAGE">
   <mimetype>image/jpeg</mimetype>
   <mimedata encoding="base64-gzip">
      H5NstasOO4H=98YDADB3jgSFg+9g813y5h9SDYG2
   </mimedata>
</data>
<data sid="INVALIDIMAGE">
   <mimetype>image/jpeg</mimetype>
   <mimedata encoding="base64-gzip">
      3y5h9SDY3jgSFg+9g81G2O4HH=98YD5NstasOADB
   </mimedata>
</data>
```

Using Entrust Signatures

The Entrust engine uses Entrust certificates rather than RSA certificates. As such, users must have Entrust software installed on their computers to access those certificates.

To create a signature button that uses the Entrust engine, set the following parameters in the *signformat* option:

- **MIME type —** The MIME type that is used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine —** The name of the signing engine to use. In this case, **Entrust**.
- **delete —** Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

For example, the generic XFDL code for a button using the Entrust engine looks like this:

```
<button sid="BUTTON1">
   <type>signature</type>
   <signformat>
      application/vnd.xfdl;
      engine="Entrust"
   </signformat>
</button>
```

Using Silanis Signatures

The Silanis engine uses special Silanis technology. As such, users must have Silanis software installed on their computers to access those certificates.

To create a signature button that uses the Silanis engine, set the following parameters in the *signformat* option:

- **MIME type** — The MIME type that is used to store the signature information. You should always use *application/vnd.xfdl*.
- **engine** — The name of the signing engine to use. In this case, **Silanis**.
- **delete** — Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures. If you want to prevent a signature from being deleted, set this to **off**.

Additionally, Silanis software allows you to create No-Lock signatures by adding the following parameter:

- **lock —** This flag sets whether the signature locks the data. By default, this is set to **on**, which means signatures lock the data they sign. If you want to prevent a signature from locking data, set this to **off**. For more information, refer to ″About No-Lock Signatures″.

Sizing the Signature Button and Scaling the Image

Silanis signatures display an image rather than text on the signature button. This allows the Silanis software to show a graphic of the user's handwritten signature once the form is signed. In most cases, this image is larger than the standard button size for a digital signature. To account for this, you should:

- Make the signature button larger than normal.
- Use the *imagemode* option to scale the image to the correct size.

To make the signature button larger, use the extent setting in the *itemlocation* option to change it's height. You may want to experiment to determine the appropriate size.

To ensure that the image fits within the button, you should also set the *imagemode* option to **scale**. This will resize the image (either by shrinking it or enlarging it) to a best fit within the button, and will maintain the original aspect ratio.

For example, the following code creates a button that is 300 pixels wide and 150 pixels tall, and that scales the image to fit the button:

```
<button sid="BUTTON1">
   <itemlocation>
      <x>10</x>
      <y>10</y>
      <width>300</width>
      <height>150</height>
   </itemlocation>
   <imagemode>scale</imagemode>
</button>
```

Storing the Signature Images

The graphic that displays the user's handwritten signature is captured once and then saved until needed. It is not placed in the form until the form is signed. However, the form must still contain a data item that can store the image after signing. This means that you must add a data item to your form for this purpose. The data item should include an empty mimedata option, as shown:

```
<data sid="SIGIMAGE">
   <mimedata></mimedata>
</data>
```

This item will store the images provided by Silanis once the form is signed.

You also need to supply a "blank" image that the button can display before the form is signed. This image does not actually have to be blank, but should indicate that the button has not yet been signed.

To add the blank image, create another data item that contains your image, as shown:

```
<data sid="BLANKIMAGE">
   <mimetype>image/jpeg</mimetype>
   <mimedata encoding="base64-gzip">
      H5NstasOO4H=98YDADB3jgSFg+9g813y5h9SDYG2
   </mimedata>
</data>
```

Note: Silanis provides a standard image that you can use for unsigned buttons. However, this image is only available through the Designer, which will automatically set up the signature button for you.

Linking to the Signature Image

As already discussed, you should include an empty data item in your form. This item stores the Silanis signature image once the form is signed. However, before the Viewer can locate that data item, you must add a link to the button that points to it.

To create this link, set the *signatureimage* option to the name of the data item. For example, to link to a data item called SIGIMAGE, you would use the following code:

```
<button sid="BUTTON1">
   <signatureimage>SIGIMAGE</signatureimage>
</button>
```

Note that this does not control which image is displayed by the button. It simply creates a link that tells the Viewer which data item contains the signature image.

Displaying the Right Image

The signature button must display one of two images: the blank image, or the signature image provided by the Silanis software. As with any button, you use the *image* option to set which image is displayed. However, in this case you must also use a compute to determine which image to display.

This compute relies on the *signer* option in the button: if the *signer* option is empty, then the form has not been signed and the *image* option should be set to the blank image; if the *signer* option has any other value, then there is a signature present and the *image* option should be set to the image supplied by Silanis.

The following example shows a compute that makes this decision. In this case, the data items that contain the images are named BLANKIMAGE and SIGIMAGE:

```
<button sid="BUTTON1">
   <image compute="signer == '' ? 'BLANKIMAGE' : &#xA;
      'SIGIMAGE')"></image>
</button>
```

Putting it All Together

The following example shows a signature button that is completely configured for a Silanis signature. In addition, the code includes the BLANKIMAGE data item and the SIGIMAGE data item:

```
<button sid="BUTTON1">
   <type>signature</type>
   <signformat>application/vnd.xfdl; engine="Silanis"
   </signformat>
   <itemlocation>
      <x>10</x>
      <y>10</y>
      <width>300</width>
      <height>150</height>
   </itemlocation>
   <imagemode>scale</imagemode>
   <signatureimage>SIGIMAGE</signatureimage>
   <image compute="signer == '' ? 'BLANKIMAGE' : &#xA;
      'SIGIMAGE')"></image>
</button>
<data sid="BLANKIMAGE">
   <mimetype>image/jpeg</mimetype>
   <mimedata encoding="base64-gzip">
      H5NstasOO4H=98YDADB3jgSFg+9g813y5h9SDYG2
   </mimedata>
</data>
<data sid="SIGIMAGE">
   <mimedata></mimedata>
</data>
```

About No-Lock Signatures

When you create a No-Lock signature, the Viewer does not lock the data covered by that signature. This means that you can make changes to the data even though it is signed. Making changes in this way still invalidates the signature, but you can then apply a second signature to the same data to approve the changes. This returns the original signature to a valid state.

This is useful in some workflows. For example, you might create a form in which an employee completes a section and then signs. You might then want a manager to review the form, make any corrections that are necessary to the employee's work, and then sign the form himself to approve the changes.

Creating a No-Lock Signature

To create a No-Lock signature, set the lock parameter to off in the *signformat* option, as shown:

```
<signformat>application/vnd.xfdl; engine="Silanis"; lock="off"
</signformat>
```

When creating a No-Lock signature, you must also add a special setting to the *value* and *signer* options for the signature button. This setting is the *transient* attribute, which is added to the *value* and *signer* tags as shown:

```
<button sid="BUTTON1">
   <value transient="on"></value>
   <signer transient="on"></signer>
</button>
```

Finally, you must add the *transient* attribute to the *image* option as well. In this case, the *transient* attribute can go either before or after the *compute* attribute. For example, the following *image* option has the *transient* attribute before the *compute* attribute:

```
<image transient="on" compute="signer == '' ? &#xA;
   'BLANKIMAGE' : 'SIGIMAGE')"></image>
```

The transient setting allows the options to temporarily change their values without breaking any signatures. This ensures that the signature does not become stuck in a particular state once a second, overlapping signature is applied.

Creating an Approving Signature for No-Lock Scenarios

An approving signature is a signature that you add to the form so that your users can approve changes made to data that is covered by a No-Lock signature. The approving signature must sign all of the data that is covered by the No-Lock signature, as well as the No-Lock signature itself. If it does not cover the same data as the No-Lock signature, the No-Lock signature will not return to a valid state when the approving signature is applied.

The approving signature may also cover additional data. For instance, you may have a ″public″ section of your form that is covered by a No-Lock signature, and a ″office use only″ section that is covered by an approving signature. In this case, the approving signature would cover both the ″public″ and ″office use only″ sections.

When you create an approving signature, you must:

- Prevent the the No-Lock signature from signing its own signature image.
- Prevent all approving signatures from signing the image for the No-Lock signature.

This ensures that the signature image can change later if the state of the signature changes. For example, if something happens to make the signature invalid, then the image must change to reflect that.

Preventing the No-Lock Signature from Signing the Image

To prevent the No-Lock signature from signing its signature image, use the *signitemrefs* filter to omit the entire data item from the signature. For example, if the signature image were stored in the SIGIMAGE1 item, you would use the following filter:

```
<signitemrefs>
   <filter>omit</filter>
   <itemrefs>SIGIMAGE1</itemrefs>
</signitemrefs>
```

If your signature item already contains a *signitemrefs* filter that omits items, simply add the signature image to the list of items.

Preventing Approving Signatures from Signing the Image

Next you need to prevent all approving signatures from signing the signature image for the No-Lock signature. Again, you use the *signitemrefs* filter to do this, as shown:

```
<signitemrefs>
   <filter>omit</filter>
   <itemrefs>SIGIMAGE1</itemrefs>
</signitemrefs>
```

You must add this filter to all signatures that will sign the No-Lock signature.

Putting it All Together

The following example shows two buttons: a No-Lock signature button, and an approving signature button (that locks the form). The No-Lock signature button stores its image in the SIGIMAGE1 data item, while the approving signature button stores its image in the SIGIMAGE2 data item.

Note that only the filters already discussed have been added to the signature buttons. In an actual form, the No-Lock signature would also filter out the approving signature.

```
<button sid="NoLockSig">
   <type>signature</type>
   <signformat>
      application/vnd.xfdl;
      engine="Silanis";
      lock="off"</signformat>
   <itemlocation>
      <x>10</x>
      <y>10</y>
      <width>300</width>
      <height>150</height>
   </itemlocation>
   <imagemode>scale</imagemode>
   <signatureimage>SIGIMAGE1</signatureimage>
   <value transient="on"></value>
   <signer transient="on"></signer>
   <image transient="on" compute="signer == '' ? &#xA;
      'BLANKIMAGE' : 'SIGIMAGE1')"></image>
   <signitemrefs>
      <filter>omit</filter>
      <itemrefs>SIGIMAGE1</itemrefs>
   </signitemrefs>
</button>
<button sid="ApprovingSig">
   <type>signature</type>
   <signformat>application/vnd.xfdl; engine="Silanis"
   </signformat>
   <itemlocation>
      <x>10</x>
      <y>40</y>
      <width>300</width>
      <height>150</height>
   </itemlocation>
```

```
      <imagemode>scale</imagemode>
      <signatureimage>SIGIMAGE2</signatureimage>
      <value></value>
      <signer></signer>
      <image compute="signer == '' ? 'BLANKIMAGE' : &#xA;
         'SIGIMAGE2')"></image>
      <signitemrefs>
         <filter>omit</filter>
         <itemrefs>SIGIMAGE1</itemrefs>
      </signitemrefs>
   </button>
   <data sid="BLANKIMAGE">
      <mimetype>image/jpeg</mimetype>
      <mimedata encoding="base64-gzip">
         H5NstasOO4H=98YDADB3jgSFg+9g813y5h9SDYG2
      </mimedata>
   </data>
   <data sid="SIGIMAGE1">
      <mimedata></mimedata>
   </data>
   <data sid="SIGIMAGE2">
      <mimedata></mimedata>
   </data>
```

Reference: Available Options for Digital Signature Buttons

The following table lists all of the options that apply to digital signature buttons:

Option	Behaviour
signature	Sets the name (sid) that is used for the *signature* item when it is created during signing.
signatureimage	Sets the name (sid) that is used for the data item that stores the signature's image. Used only with Silanis signatures.
signdatagroups	Sets which datagroups will be included or excluded from the signature.
signer	Records the identity of the person who signed the form.
signformat	Sets the details of the signature, including the MIME type to encode it, the signature engine to create it, and special settings for the signature engine.
signgroups	Sets which groups will be included or excluded from the signature.
signinstance	Sets which XForms instance data will be included or excluded from the signature.
signitemrefs	Sets specific items that will be included or excluded from the signature.
signitems	Sets which items will be included or excluded from the signature.
signnamespaces	Sets which namespace elements and attributes will be included or excluded from the signature.
signoptionrefs	Sets specific options that will be included or excluded from the signature.
signoptions	Sets which options will be included or excluded from the signature.
signpagerefs	Sets specific pages that will be included or excluded from the signature.

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.

© Copyright IBM Corp. 2003, 2006

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

S325-2604-00