PureEdge

PUREEDGE SOLUTIONS INC.

XFDL

Specification v6.0

Internet Commerce System™

Editors:

John Boyer (PureEdge), jboyer@pureedge.com

Tim Bray (Textuality), tbray@textuality.com

Contributors:

Richard Bennett (PureEdge)

David Manning (PureEdge)

Michael Mansell (PureEdge)

This version:

Revision 3, August 6, 2003.

Copyright:

Copyright © 2000 – 2003 by PureEdge Solutions Incorporated. All rights reserved.

U.S. Government Restricted Rights. The Product is provided with RESTRICTED RIGHTS. Use, modification, reproduction, release, performance, display or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of The Rights in Technical Data - Noncommercial Items, DFARS 252.227-7013. The rights stated in subparagraph (c) of the Commercial Computer Software - Restricted Rights, 48 CFR 52.227-19 are hereby withheld from the government. Manufacturer is PureEdge Solutions Inc., located at 4396 West Saanich Rd., Victoria, BC Canada V8Z 3E9.

PureEdge Solutions is a trademark of PureEdge Solutions Incorporated. All other products or services mentioned in this manual are trademarks, registered trademarks, service marks, or registered service marks of their respective companies or organizations.

PureEdge Solutions **www.PureEdge.com**

tel. 1-888-517-2675 **fax.** 250-708-8010 **email.** info@PureEdge.com

address. 4396 West Saanich Rd., Victoria, B.C. V8Z 3E9

Contents

1. INTRODUCTION 1

1.1 ORIGIN AND GOALS 1
1.2 REFERENCES 2
1.3 TERMINOLOGY 2
1.4 NOTATION 2
1.5 OVERLAP WITH OTHER SPECIFICATIONS 2
1.6 NON-REPUDIATION AND THE DOCUMENT-CENTRIC MODEL 3
1.7 FORM NAMES AND EXTENSIONS 5
1.8 MIME TYPES 5

2. THE STRUCTURE OF XFDL FORMS 7

2.1 TOP-LEVEL STRUCTURE 7
2.2 ITEMS 8
2.3 OPTIONS AND ARRAY ELEMENTS 11
2.4 RESTRICTED CHARACTERS IN CHARACTER DATA CONTENT 17
2.5 BASE-64 AND COMPRESSED ENCODING OF BINARY DATA 18
2.6 SCOPE IDENTIFIERS (SID) 18
2.7 DOCUMENT REPRODUCIBILITY 18

3. THE XFDL COMPUTE SYSTEM 21

3.1 WHITESPACE IN COMPUTES 21
3.1.1 Multiline Computes 21
3.2 STRUCTURE OF MATHEMATICAL AND CONDITIONAL EXPRESSIONS 22
3.3 TABLE OF OPERATORS 23
3.3.1 Precedence of Operations 24
3.3.2 Decision Operations and Namespace Qualification 25
3.3.3 Illegal Characters in XML Attributes 25
3.4 DEFINITION OF VALUE 25
3.5 QUOTED STRINGS 26
3.6 XFDL REFERENCES TO ELEMENTS 27
3.6.1 Referencing the XFDL Version 29
3.7 FUNCTION CALL SYNTAX 29
3.8 REPRESENTING AND RUNNING XFDL COMPUTES 29
3.8.1 Introduction 29
3.8.2 Cached Dependency Lists 30
3.8.3 Topological Sorting 32
3.8.4 Handling Dynamic References 33
3.8.5 Reference Caching 34
3.8.6 Re-entrancy 34

3.8.7 Duplicate Entries on the Total Change List . . . 35
3.8.8 Missing References . . . 35
3.8.9 Handling of Element Deletion . . . 35
3.8.10 Limitations . . . 35

4. SIGNATURES IN XFDL . . . 37

4.1 Applying Signature Filters . . . 37
4.2 Namespaces in Signature Filters . . . 38
4.3 Applying Multiple Signatures . . . 39
4.4 Securing Signed Elements . . . 39
4.5 Preventing Layout Changes . . . 39
4.6 Preventing Exploitable Overlaps of Signed Elements . . . 40

5. SMALL XFDL FORM EXAMPLES . . . 43

6. GLOBAL SETTINGS . . . 47

6.1 Form Globals . . . 47
6.2 Page Globals . . . 49

7. DETAILS ON ITEMS . . . 51

7.1 ACTION . . . 52
7.2 BOX . . . 53
7.3 BUTTON . . . 54
7.4 CELL . . . 57
7.5 CHECK . . . 59
7.6 COMBOBOX . . . 60
7.7 DATA . . . 63
7.8 FIELD . . . 64
7.9 HELP . . . 66
7.10 LABEL . . . 67
7.11 LINE . . . 69
7.12 LIST . . . 70
7.13 POPUP . . . 73
7.14 RADIO . . . 75
7.15 SIGNATURE . . . 77
7.16 SPACER . . . 80
7.17 TOOLBAR . . . 81
7.18 <CUSTOM ITEM> . . . 82

8. DETAILS ON OPTIONS AND ARRAY ELEMENTS . . . 83

8.1 ACCLABEL . . . 86
8.2 ACTIVATED . . . 88
8.3 ACTIVE . . . 90
8.4 BGCOLOR . . . 91

8.5 BORDERWIDTH . . . 93
8.6 COORDINATES . . . 94
8.7 DATA . . . 95
8.8 DATAGROUP . . . 96
8.9 DELAY . . . 98
8.10 DIRTYFLAG . . . 100
8.11 EDITSTATE . . . 101
8.12 EXCLUDEDMETADATA . . . 102
8.13 FILENAME . . . 104
8.14 FOCUSED . . . 105
8.15 FOCUSEDITEM . . . 107
8.16 FONTCOLOR . . . 108
8.17 FONTINFO . . . 110
8.18 FORMAT . . . 112
8.19 FORMID . . . 120
8.20 FULLNAME . . . 122
8.21 GROUP . . . 123
8.22 HELP . . . 125
8.23 IMAGE . . . 126
8.24 IMAGEMODE . . . 127
8.25 ITEMFIRST . . . 129
8.26 ITEMLAST . . . 130
8.27 ITEMLOCATION . . . 131
8.28 ITEMNEXT . . . 140
8.29 ITEMPREVIOUS . . . 141
8.30 JUSTIFY . . . 142
8.31 KEYPRESS . . . 143
8.32 LABEL . . . 145
8.33 LABELBGCOLOR . . . 146
8.34 LABELBORDERWIDTH . . . 148
8.35 LABELFONTCOLOR . . . 149
8.36 LABELFONTINFO . . . 151
8.37 LAYOUTINFO . . . 153
8.38 LINESPACING . . . 155
8.39 MIMEDATA . . . 156
8.40 MIMETYPE . . . 159
8.41 MOUSEOVER . . . 160
8.42 NEXT . . . 162
8.43 PAGEFIRST . . . 163
8.44 PAGEID . . . 164
8.45 PAGELAST . . . 165
8.46 PAGENEXT . . . 166
8.47 PAGEPREVIOUS . . . 167
8.48 PREVIOUS . . . 168
8.49 PRINTBGCOLOR . . . 169

8.50 PRINTFONTCOLOR . . . 171
8.51 PRINTLABELBGCOLOR . . . 173
8.52 PRINTLABELFONTCOLOR . . . 175
8.53 PRINTSETTINGS . . . 177
8.54 RTF . . . 180
8.55 SAVEFORMAT . . . 182
8.56 SCROLLHORIZ . . . 185
8.57 SCROLLVERT . . . 186
8.58 SIGNATURE . . . 187
8.59 SIGNATUREIMAGE . . . 188
8.60 SIGNDATAGROUPS . . . 189
8.61 SIGNDETAILS . . . 191
8.62 SIGNER . . . 196
8.63 SIGNFORMAT . . . 198
8.64 SIGNGROUPS . . . 207
8.65 SIGNITEMREFS . . . 209
8.66 SIGNITEMS . . . 211
8.67 SIGNNAMESPACES . . . 213
8.68 SIGNOPTIONREFS . . . 215
8.69 SIGNOPTIONS . . . 217
8.70 SIGNPAGEREFS . . . 219
8.71 SIZE . . . 221
8.72 TEXTTYPE . . . 223
8.73 THICKNESS . . . 224
8.74 TRANSMITDATAGROUPS . . . 225
8.75 TRANSMITFORMAT . . . 227
8.76 TRANSMITGROUPS . . . 231
8.77 TRANSMITITEMREFS . . . 233
8.78 TRANSMITITEMS . . . 235
8.79 TRANSMITNAMESPACES . . . 237
8.80 TRANSMITOPTIONREFS . . . 239
8.81 TRANSMITOPTIONS . . . 241
8.82 TRANSMITPAGEREFS . . . 243
8.83 TRIGGERITEM . . . 245
8.84 TYPE . . . 246
8.85 URL . . . 249
8.86 VALUE . . . 253
8.87 VISIBLE . . . 255
8.88 <CUSTOM OPTION> . . . 256

9. FUNCTION CALLS . . . 257

9.1 STRING FUNCTIONS . . . 259
9.1.1 countLines . . . 259
9.1.2 countWords . . . 261
9.1.3 pad . . . 262

9.1.4 replace . . . 264
9.1.5 strlen . . . 266
9.1.6 strmatch . . . 267
9.1.7 strpbrk . . . 269
9.1.8 strrstr . . . 270
9.1.9 strstr . . . 271
9.1.10 substr . . . 272
9.1.11 tolower . . . 274
9.1.12 toupper . . . 275
9.1.13 trim . . . 276
9.1.14 URLDecode . . . 277
9.1.15 URLEncode . . . 278
9.2 MATH FUNCTIONS . . . 279
9.2.1 abs . . . 279
9.2.2 acos . . . 280
9.2.3 annuity . . . 281
9.2.4 asin . . . 283
9.2.5 atan . . . 284
9.2.6 ceiling . . . 285
9.2.7 compound . . . 286
9.2.8 cos . . . 287
9.2.9 deg2rad . . . 288
9.2.10 exp . . . 289
9.2.11 fact . . . 290
9.2.12 floor . . . 291
9.2.13 ln . . . 292
9.2.14 log . . . 293
9.2.15 mod . . . 294
9.2.16 pi . . . 295
9.2.17 power . . . 296
9.2.18 rad2deg . . . 297
9.2.19 rand . . . 298
9.2.20 round . . . 299
9.2.21 sin . . . 300
9.2.22 sqrt . . . 301
9.2.23 tan . . . 302
9.3 UTILITY FUNCTIONS . . . 303
9.3.1 applicationName . . . 303
9.3.2 applicationVersion . . . 304
9.3.3 applicationVersionNum . . . 305
9.3.4 checkValidFormats . . . 306
9.3.5 countDatagroupItems . . . 308
9.3.6 countGroupedItems . . . 309
9.3.7 decimal . . . 311
9.3.8 destroy . . . 312

9.3.9 duplicate . . . 313
9.3.10 formatString . . . 315
9.3.11 generateUniqueName . . . 317
9.3.12 get . . . 319
9.3.13 getAttr . . . 320
9.3.14 getGroupedItem . . . 322
9.3.15 getPosition . . . 324
9.3.16 getPref . . . 325
9.3.17 getReference . . . 328
9.3.18 isValidFormat . . . 330
9.3.19 set . . . 332
9.3.20 setAttr . . . 334
9.3.21 toggle . . . 336
9.3.22 xmlmodelUpdate . . . 338
9.4 Time and Date Functions . . . 340
9.4.1 date . . . 340
9.4.2 dateToSeconds . . . 341
9.4.3 day . . . 343
9.4.4 dayOfWeek . . . 344
9.4.5 endOfMonth . . . 345
9.4.6 hour . . . 346
9.4.7 minute . . . 347
9.4.8 month . . . 348
9.4.9 now . . . 349
9.4.10 second . . . 350
9.4.11 time . . . 351
9.4.12 year . . . 352

APPENDIX A: QUICK REFERENCE TABLES . . . 353

Table of Items and Form and Page Globals . . . 353
Table of Options . . . 356
Cross Reference Table for Items and Options . . . 376

APPENDIX B: DEFAULT SIZES . . . 379

APPENDIX C: ORDER OF PRECEDENCE OF FILTERS . . . 381

APPENDIX D: COLOR TABLE . . . 387

INDEX . . . 397

1. Introduction

This document describes a class of XML documents called Extensible Forms Description Language (XFDL) Forms and partially describes the behavior of computer programs that process them. An XFDL processor is a software program that reads, processes, and writes XFDL forms. Processing may include such tasks as GUI rendering, data extraction, or modification.

1.1 Origin and Goals

From 1993 to 1998, PureEdge developed the Universal Forms Description Language (UFDL). XFDL is the result of developing an XML syntax for the UFDL, thereby permitting the expression of powerful, complex forms in a syntax that promotes application interoperability and adherence to worldwide Internet standards. The current design goals of XFDL are to create a high-level computer language that:

1. Represents forms as single objects without dependencies on externally defined entities, thus allowing them to act as contractual documents.
2. Is represented by human-readable plain text.
3. Is a publicly accessible open standard.
4. Provides a syntax for inline mathematical and conditional expressions.
5. Permits the enclosure of an arbitrary size and number of base-64 encoded binary files.
6. Offers precision layout needed to represent and print dense business/government forms.
7. Facilitates server-side processing via client-side input validation and formatting.
8. Permits extensibility including custom items, options, and external code functions.
9. Offers comprehensive signature support, including:
   - Capture of the whole context of a business transaction
   - Multiple signers
   - Different signers of (possibly overlapping) portions of a form
   - Freezing computations on signed portions of a form

Maintaining the data, logic, and presentation layers in a single, legally binding document is a paradigm shift in electronic commerce, which traditionally separates each layer. The decision to use a more “document-centric” model was not made lightly — it was found to be necessary in order to provide legal non-repudiation.

1.2 References

[1] Boyer, J. Lexical and Syntactic Specification for the Universal Forms Description Language (UFDL) Version 4.0. PureEdge Solutions. 6 SEP 1997.

[2] Bray, T., Paoli, J. & Sperberg-McQueen, C.M. (Eds.) Extensible Markup Language (XML) 1.0. W3C Recommendation. http://www.w3.org/TR/1998/REC-xml-19980210.html. 10 FEB 1998.

[3] Gordon, M. (Ed.) UFDL v4.0.1 Specification. PureEdge Solutions. 1993-1998.

1.3 Terminology

Terms are defined in Section 1.2 of the XML specification (see Reference [2] above).

1.4 Notation

XFDL forms are XML documents; the form definition is encoded using XML elements and attributes. In addition, XFDL imposes many constraints on the contents of the elements and the values of the attributes. In this specification, the nesting and sequence relationships between the elements and attributes are given, where possible, in DTD notation, while the constraints on certain attribute values are given in the BNF notation found in the XML specification. The DTD-syntax description of the elements and attributes is “almost complete” in that it illustrates XFDL constructs but not additional markup variations allowed by XML-related standards (e.g. namespace declarations and interspersed elements or attributes in other namespaces). Furthermore, the content models of some XFDL elements depends on an attribute value, which is also not expressible using DTDs.

1.5 Overlap With Other Specifications

To serve its purpose, XFDL requires comprehensive presentation control and data typing machinery. This document describes a set of elements and attributes that meet these requirements. It may be the case that the presentation controls can be replaced by a W3C-specified set of form controls; however, existing specifications allow too much user interface flexibility to be suitable for security and non-repudiation purposes. Similarly, XFDL’s data typing controls could be replaced by XML Schema constructs, but the XFDL formatting system remains more functionally useful for forms applications (e.g. the ability to define user-friendly replacements for validated input).

1.6 Non-Repudiation and the Document-Centric Model

A digital signature attached to a file accurately identifies the individual who used it, based on the digital certificate provider's security and the security of the user's hardware. However, to provide full non-repudiation and auditability, a business transaction not only needs to be signed by someone whose identity is verifiable, it also needs to be representative of the context in which it was signed.

With paper-based forms and documents, this is easily accomplished. Everything that appears on the signed document is considered part of the transaction. Electronic forms and documents, however, present a more complex problem in that the exact appearance and functionality of the document must be signed as well as the user's input, or the transaction is meaningless. Legal standards for font size and color must also be observed both when the document is signed and when it is subsequently examined.

Digital signature technology alone can provide the first part of the solution, but not the second. According to the Association for Information and Image Managements' Performance Guidelines for the Legal Acceptance of Records Produced by Information Technology Systems, the only way in which an electronic document can be considered to provide non-repudiation and auditability is if it contains the following elements, clearly recognizable, in one file:

- Individual letters, numbers and symbols
- Combinations of letters, numbers and symbols forming words or sentences
- Graphics, such as signatures, logos, pictures, and so on.
- Sounds
- Other features of records such as color, shape, texture, and so on, that relate to the content of the information

XFDL can be used to create forms that meet the above criteria by presenting a business transaction as a single entity, which is updated as the user fills it in. Item values are stored within the items themselves, and the layout is retained throughout the process.

The following example shows how a section of a form's underlying XFDL code changes as a user fills it in.

This is how the XFDL code appears before the user fills in the form:

```
<field sid="PHONE_NUMBER">
   <itemlocation>
      <ae>after</ae>
      <ae>PHONE_PROMPT</ae>
   </itemlocation>
   <label>Please enter your phone number</label>
   <labelfontinfo>
      <ae>Times</ae>
      <ae>14</ae>
      <ae>bold</ae>
   </labelfontinfo>
   <value></value>
```

```
        <fontinfo>
            <ae>Times</ae>
            <ae>14</ae>
            <ae>bold</ae>
        </fontinfo>
    </field>
```

This sample illustrates a user-input field. This field is identified as appearing on the form after another item labelled “PHONE_PROMPT”. The field contains a built-in label which displays the prompt “Please enter your phone number” in a Times, 14 point, bold font.

As the user fills in the form, the values of the input items change. After the user enters a phone number, the XFDL code looks like this:

```
    <field sid="PHONE_NUMBER">
        <itemlocation>
            <ae>after</ae>
            <ae>PHONE_PROMPT</ae>
        </itemlocation>
        <label>Please enter your phone number</label>
        <labelfontinfo>
            <ae>Times</ae>
            <ae>14</ae>
            <ae>bold</ae>
        </labelfontinfo>
        <value>(250) 555-1234</value>
        <fontinfo>
            <ae>Times</ae>
            <ae>14</ae>
            <ae>bold</ae>
        </fontinfo>
    </field>
```

As this example shows, all the information describing the field’s appearance and location is still present. The value option, however, now holds the user’s phone number. This, too, is displayed in a Times, 14 point, bold font.

When the user signs the form with a digital signature, the underlying XFDL code is encrypted and stored within the form, and the user’s signature is attached. Subsequently, when the form is open in a form viewing or processing application, the encrypted data is compared against the form’s current XFDL code. If any discrepancies exist, the signature is flagged as invalid, and the form no longer provides non-repudiation or auditability. Attachments can also be included as mimedata in the form, thus enabling the user to sign both the attachments and the form itself.

This method of representing and collecting information in forms and digitally signing and encrypting them ensures that the identity of the signer can be confirmed and that the signer can be proven to have signed the full content and context of the form.

Note: *The presentation layer should be augmented by an XML data model (see Section ??) to increase long-term maintainability and interoperability of forms applications.*

1.7 Form Names and Extensions

To ensure cross-platform compatibility, form names should avoid the use of characters that are illegal under popular operating systems or in URLs. For names should also include the .xfdl extension. The extension .xfd can also be used, but it should be avoided if possible.

1.8 MIME Types

XFDL supports the following MIME types:

- application/vnd.xfdl

2. The Structure of XFDL Forms

2.1 Top-Level Structure

An XFDL form is an XML 1.0 document whose root element type is XFDL. This element must be in the XFDL namespace, a URI that includes the major and minor version of XFDL. For example,

```
[1]  <XFDL xmlns="http://www.PureEdge.com/XFDL/6.0"> ... </XFDL>
```

The XFDL element must contain a <globalpage> element as the first child element, followed by one or more <page> elements.

```
[2]  <!ELEMENT XFDL (globalpage, page+)>
```

The <globalpage> element must contain a single <global> element, which can contain zero or more option elements. These are referred to as form global options; they typically contain information applicable to the whole form or default settings for options appearing in the element content of pages. The <globalpage> and <global> elements must contain an attribute called *sid* which must be set to the value global. Although the attribute has a fixed value, it is still required because XFDL processors must be able to clearly identify global objects by sid even in the presence of interspersed custom elements in non-XFDL namespaces.

```
[3]  <!ELEMENT globalpage (global)>
[4]  <!ELEMENT global (%options;*)>
[5]  <!ATTLIST globalpage sid CDATA #REQUIRED #FIXED "global">
     <!-- This rule is only intended to communicate the  restriction with DTD-like
     notation; DTDs don't allow required and fixed (XML Schema does) -->
```

A <page> element contains a <global> element followed by zero or more 'item' elements. The options in the page's global element typically contain information applicable to the whole page or default settings for options appearing within element content of items. The page global options take precedence over form global options. A page is also required to have a 'sid' attribute, which provides an identifier that is unique among all <page> elements (sid is short for scope identifier). The 'sid' attribute

value must not be the word ‘global’ and is otherwise a letter followed by any combination of zero or more letters, digits and underscores.

```
[6]  <!ELEMENT page (global, %items;*)>
[7]  <!ATTLIST page sid CDATA #REQUIRED>
[8]  sid ::= (Letter | Special) (Letter | Special | Digit | ‘_’)* - (‘global’)
[9]  Letter ::= [A-Z] | [a-z]
[10] Digit ::= [0-9]
[11] Special ::= [0x00C0-0x00FF] - (0x00D7 | 0x00F7)
```

The intention of using multiple pages in a form is to show the user one page at a time. Each page should contain items that describe GUI widgets which allow users to switch to different pages without contacting a server program. XFDL allows the page switching items to be defined in the form so the form developer can add computations that control the flow of pages based on context.

2.2 Items

An item is a single object in a page of a form. Some items represent GUI widgets, such as buttons, check boxes, popup lists, and text fields. Other items are used to carry information such as an attached word processing document, a signature, or application-specific job descriptions (e.g. workflow or database requests).

An item can contain zero or more option elements. The options define the characteristics of the item. An item with zero options is completely defined by the option defaults. Each item must have a sid attribute.

```
[12] <!ELEMENT %items; (%options;*)>
[13] <!ATTLIST %items; sid CDATA #REQUIRED>
```

The parameter entity reference to “%item;” could be defined partially as:

```
[14] <!ENTITY % item "(action | box | button | cell | check | combobox | data | field
     | help | label | line | list | popup | radio | signature | spacer | toolbar)">
```

The details of each type of item listed in the rule above are discussed in “Details on Items”on page 49, but is summarized here for your convenience.

action

A non-visible item that can perform similar tasks to a button (print, cancel, submit, and so on) either after a certain period of time or with a regular frequency.

box

A graphic effect that is typically used to visually group a set of other GUI widgets on the page. A box is drawn under all widgets on a page.

button

Performs one of a variety of tasks when pressed by the user, such as saving, printing, canceling, submitting, digitally signing the form, viewing documents enclosed in the form, and so on. A button can have a text or image face.

cell

Defines a single entry in a list, popup or combobox; selecting a cell can change the value of the associated list-type item, but it can also perform many of the actions of a button, such as submitting, printing, saving, and so on.

check

Defines a single check box.

combobox

An edit field combined with a popup list; its value can be either selected or typed.

data

Used to carry binary information using base-64 encoding, such as enclosed files or digital images, using base-64 encoding.

field

Used to capture single- or multiple-line textual input from the user; it includes input validation features.

help

Carries text information that can be associated with one or more other items; the user can read this text by utilizing the page's help feature.

label

Shows either an image or a single or multiple line text value.

line

A simple graphic effect used as a separator.

list

Shows a list box populated with cell items from which the user can choose one.

popup

Shows either the text of the currently selected cell or a label if there is no selection; the popup provides a small button that causes the list of selectable cells to appear.

radio

Provides a single radio button; it is associated with other radio buttons to form a group, and only one radio button can be on in a group.

signature

Receives the signature that ultimately results when a user presses a signature button.

spacer

An invisible GUI widget that facilitates spacing in the relational positioning scheme.

toolbar

Items associated with a toolbar item appear in a separate window pane above the pane for the form page; it is the typical location for page switching and other buttons as its contents are not printed if the form is rendered on paper.

Note: *The parameter entity %items is not intended as a formal definition of the content model of a page (after the global element). It is only intended to present the list of items. The items could be explicitly namespace qualified with a prefix bound to the XFDL URI (given in Rule 1). Moreover, XFDL permits form authors to intersperse custom elements among the items as long as those elements have a 'sid' attribute in the XFDL namespace. Custom elements can be used, for example, to carry complex instructions and logic for server-side components. While simple static application-specific information could be represented with XML processing instructions, many server side applications (e.g. workflow and database requests) require complex instructions that can include the use of the XFDL compute system to collect information from around the form. For more information, see section "7.18 <custom item>".*

2.3 Options and Array Elements

Options can appear as form globals, page globals, or as the contents of items. An option defines a named property of an item, page, or form. The parameter entity reference to “%option;” could partially be defined as follows:

```
[15] <!ENTITY % option "(acclabel | activated | active | bgcolor | border | coordinates | data |
     datagroup | delay | dirtyflag | editstate | excludedmetadata | filename | focused | focuseditem |
     fontcolor | fontinfo | format | formid | fullname | group | help | image | imagemode | itemlocation
     | justify | keypress | label | labelbgcolor | labelborderwidth | labelfontcolor | labelfontinfo |
     layoutinfo | linespacing | mimedata | mimetype | mouseover | next | pageid | previous |
     printbgcolor | printfontcolor | printlabelbgcolor | printlabelfontcolor | printsettings | rtf |
     saveformat | scrollhoriz | scrollvert | signature | signatureimage | signdatagroups | signdetails |
     signer | signformat | signgroups | signitemrefs | signitems | signnamespaces | signoptionrefs |
     signoptions | signpagerefs | size | texttype | thickness | transmitdatagroups | transmitformat |
     transmitgroups | transmititemrefs | transmititems | transmitnamespaces | transmitoptionrefs |
     transmitoptions | transmitpagerefs | triggeritem | type | url | value | version | visible)">
```

Again, the definition is partial because XFDL supports namespace qualification of options as well as the interspersion of custom options in non-XFDL namespaces. Typically, application-defined options occur in application-defined items, but they are also sometimes used in XFDL-defined items to store intermediate results of complex computations, thereby allowing the form developer to arbitrarily break down a problem into manageable pieces. For more information, see section “8.88 <custom option>”.

The XFDL options are fully discussed in section “8. Details on Options and Array Elements” and are summarized here:

acclabel

Provides a special description of input items that is read by screen reading software.

activated, focused, and mouseover

Are not usually declared by the form developer. Instead, they are set for each item by forms viewer software based on system events. Under certain circumstances activated and focused can be set by the form developer. See the option descriptions for more information.

active

Specifies whether an item is active or inactive.

bgcolor, fontcolor, labelbgcolor, and labelfontcolor

Specify the colors for an item or its label using either predefined names or RGB triplets.

border and labelborderwidth

Control whether an item or its label has a border.

coordinates

Receives the location of a mouse click on an image, if the image is in a button.

data and datagroup

Used to create an association between data items and the buttons that provide file enclosure functionality.

delay

Used in an action item to specify the timing for the event and whether it should be repeated.

dirtyflag

Alerts the form viewing program to the changed status of the form.

editstate

Specifies whether the item is read only, read/write, or, for single line fields, write only.

excludedmetadata

Used to store special information that is automatically excluded from signatures.

filename and mimetype

Give additional information about an enclosed document.

focuseditem

Records which item currently has the focus.

fontinfo and labelfontinfo

Defines the typeface, point size, and special effects (bold, italics, underline and character set) for the font used to display the item's value or label.

format

Contains sub-elements that parameterize input validation for the item's value.

formid

Defines a unique identifier for the form, such as a serial number.

fullname, layoutinfo, signature, signatureimage, signdatagroups, signdetails, signer, signformat, signgroups, signitemrefs, signitems, signnamespaces, signoptionrefs, signoptions, and signpagerefs

Work together to provide a full-featured signature as defined in section “1.1 Origin and Goals” (goal 9).

group

Associates radio buttons together, and it associates cells with lists, popups, and comboboxes.

help

Identifies the help item associated with the item.

image

Identifies the data item containing the image for the button or label.

imagemode

Specifies the display behavior of the image within the data item; the image may be clipped, resized, or scaled to fit the item.

itemlocation, size and thickness

Help to define the location and size of the item.

justify

Controls whether text in the item should be left, center, or right justified.

keypress

Records the last key pressed by the user and propagates it upwards in the hierarchy.

label

Associates a simple text label with the item; labels can also be created independently with a label item.

linespacing

Adjusts the spacing between lines of text in an item.

mimedata

Used to store large binary data blocks encoded in base-64 format.

next and previous

Link the item into the tab order of the page.

pageid

Defines a unique identifier for a page, such as a serial number.

printbgcolor, printlabelbgcolor, printfontcolor, and printlabelfontcolor

Provide the ability to set printing colors for each indicated option different from the display colors on the screen.

printsettings

Parameterizes the paper rendition of a form.

rtf

Contains the rich text value of rich text fields.

saveformat and transmitformat

Control how the form is written (XFDL, HTML) when it is saved or submitted.

scrollhoriz and scrollvert

Control whether a text field item has horizontal and vertical scrollbars or whether it wordwraps, allows vertical sliding, and so on.

texttype

Sets whether a field contains plain text or rich text.

transmitdatagoups, transmitformat, transmitgroups, transmititiemrefs, transmititems, transmitnamespaces, transmitoptionrefs, transmitoptions, and transmitpagerefs

Work together to allow you to transmit form submissions.

triggeritem

Set in the form globals to identify which action, button, or cell activated a form transmission or cancellation.

type

Specifies whether the action, button, or cell item will perform a network operation, print, save, digitally sign, and so on.

url

Provides the address for a page switch, or for a network link or submission.

value

Holds the primary text associated with the item.

version

Defines the version of XFDL used to write the form.

visible

Determines whether the item should be shown to the user or made invisible.

The content of an option can take one of two formats: simple character data or a subtree of XML elements. In the latter case, the element children are typically referenced as a zero-based array. The XFDL-specific options that have simple content versus array content are defined by the parameter entity references %simpleOption; and %arrayOption; below:

```
[16] <!ENTITY % simpleOption "(acclabel | activated | active | border | data | dirtyflag | editstate |
     filename | focused | focuseditem | formid | fullname | group | help | image | imagemode | justify |
     keypress | label | labelborderwidth | linespacing | mimedata | mimetype | mouseover | next |
     pageid | previous | rtf | saveformat | scrollhoriz | scrollvert | signature | signatureimage | signer |
     signformat | texttype | thickness | transmitformat | triggeritem | type | value | version | visible)">

[17] <!ENTITY % arrayOption "(bgcolor | coordinates | datagroup | delay | excludedmetadata |
     fontcolor | fontinfo | format | itemlocation | labelbgcolor | labelfontcolor | labelfontinfo |
     layoutinfo | printbgcolor | printfontcolor | printlabelbgcolor | printlabelfontcolor | printsettings |
     signdatagroups | signdetails | signgroups | signitemrefs | signitems | signnamespaces |
     signoptionrefs | signoptions | signpagerefs | size | transmitdatagroups | transmitgroups |
     transmititemrefs | transmititems | transmitnamespaces | transmitoptionrefs | transmitoptions |
     transmitpagerefs | url)">
```

As shown below, the content model of simple options is character data. The interpretation of character data content of simple options can be qualified by an *encoding* attribute. Except for the <mimedata> option, the default encoding of simple content is ‘xml’, which means that the character data should be interpreted as plain text. The default for the mimedata option is base64-gzip. A content encoding of base64 can also be used (for example if the content is a compressed binary image such as a jpeg). If the encoding is not ‘xml’ then an XFDL-compliant processor applies the requisite decoders before interpreting the content in the application context, and it applies the appropriate encoders during

serialization. The XFDL encoding attribute can also be used on custom options, but it would have to be namespace qualified since custom options are in a non-XFDL namespace.

[18] <!ELEMENT %simpleOption; (#PCDATA)>

[19] <!ATTLIST mimedata encoding (xml | base64 | base64-gzip) "base64-gzip">

[20] <!ENTITY % plainTextOption "(acclabel | activated | active | border | data | dirtyflag | editstate | filename | focused | focuseditem | formid | fullname | group | help | image | imagemode | justify | keypress | label | labelborderwidth | linespacing | mimetype | mouseover | next | pageid | previous | rtf | saveformat | scrollhoriz | scrollvert | signature | signatureimage | signer | signformat | texttype | thickness | transmitformat | triggeritem | type | value | version | visible)">

[21] <!ATTLIST %plainTextOption; encoding (xml | base64 | base64-gzip) "xml">

The content of simple options can also be qualified by a *compute* attribute, which defines an expression used to obtain and update the option's content based on content elsewhere in the form. Before delving into the details of computes, the basic idea can be gleaned from the following example, which shows three user input field items. The first two show the state after the user has entered values, and the third field value shows the resulting content update based on the compute expression:

```
<field sid="FIELD1">
   <value>3</value>
</field>
<field sid="FIELD2">
   <value>4</value>
</field>
<field sid="FIELD3">
   <value compute="FIELD1.value + FIELD2.value">7</value>
</field>
```

It is typical to have a form run its computes on a client machine, then have server modules simply read the current values without parsing or running the computes. The rule below show that the compute attribute can appear on XFDL's simple options and that its basic content model is character data. See section "3. The XFDL Compute System" for details on the compute expression syntax.

[22] <!ATTLIST %simpleOption; compute CDATA #IMPLIED>

Array options contain one or more child elements, optionally separated by whitespace. The elements in an array option can be given any name, except that the element tag <ae> is reserved to mean an unnamed array element. XFDL typically does not assign names to the array elements, so the default tag

name of <ae> is most often used. The parameter entity reference %arrayElement; lists the array element tag names formally recognized by XFDL:

```
[23] <!ELEMENT %arrayOption; (%arrayElement;+)>

[24] <!ENTITY % arrayElement "(ae | copies | dialog | dialogcolumns | filteridentity | length |
     message | orientation | pagehashes | pages | printpages | range | servernotarizations | tag |
     template | value)">
```

Additionally, the children of pagehashes can have arbitrary names.

An example of an option that uses array element depth is bgcolor:

```
<bgcolor>
   <ae>255</ae>
   <ae>248</ae>
   <ae>220</ae>
</bgcolor>
```

The content models of array elements recursively follow the basic content model for an option. For array elements with simple character data content, the *encoding* and *compute* attributes can be used, and array elements support the element content model to arbitrary depth.

2.4 Restricted Characters in Character Data Content

The ampersand (&) and left angle bracket (<) are not permitted in character data content of XML elements, since these characters mark the beginning of entity references and XML tags, respectively. There may be occasions in which a developer needs to include these in XFDL character content (for example, email URLs). In such cases, the developer can include the ampersand and left angle bracket in a few different ways, such as using the XML entities & and < or simply wrapping the character content in a CDATA section. The character sequence]]>, which normally cannot appear in character data content, appears in a CDATA section as the delimiter token that marks the end of a CDATA section. When the sequence]]> appears as character data, then the right angle bracket (>), if necessary, should be expressed with an entity reference such as *>*.

These rules apply to all XFDL elements, but they are often used when constructing email URLs, which use the ampersand to indicate additional parameters. For example:

```
<url>
   <ae>
      <![CDATA[mailto:john@acme.com?&subject=Hello&
      body=Hello.+How+are+you?]]>
   </ae>
</url>
```

2.5 Base-64 and Compressed Encoding of Binary Data

In XFDL, option and suboption elements are allowed to store base-64 encoded and compressed base-64 encoded binary data such as signatures, images, enclosed word processing or spreadsheet documents, and so on. XFDL allows an *encoding* attribute to control whether an element contains data in a format other than plain XML character data. Both compressed and uncompressed base-64 encoding use no characters that are illegal in character data. Typically, based-64 and compressed base-64 encodings are used with the <mimedata> option in XFDL.

Since binary data tends to be long, XFDL processors are expected to “pretty print” the lines of base-64 (whether or not the binary data is first compressed). Each level of element depth beyond the root <XFDL> tag corresponds to three spaces. The first line of base-64 is expected to be on the line after the element start tag, and the lines of base-64 are expected to be indented three spaces more than the element start tag. For example, the <mimedata> option’s start tag is indented 9 spaces (three for the page level, three for the item level and three for the option level), so each line of base-64 content in a <mimedata> option begins with 12 spaces. This should be followed by 76 characters of base-64 (except for the last line), then a linefeed (not a return-newline sequence). Thus, the element end tag is on the line after the last line of base-64 content. The end tag is indented so that it aligns under the start tag (e.g. indent 9 spaces for the <mimedata> option).

The above method describes the creation of new base-64 encoded content. However, the normal XML preservation of whitespace in element content should be used when reading XFDL forms so that prior whitespace indenting techniques used with older XFDL forms will continue to function properly (esp. with regard to digital signature validation).

2.6 Scope Identifiers (sid)

An XFDL scope identifier, or sid, uniquely identifies an element within the scope of its logical parent. Each <page> element must have a *sid* attribute that uniquely identifies the page within the surrounding XFDL form element. An item element must have a *sid* attribute that uniquely identifies the item within the surrounding page element.

In XFDL, each option element is defined to be uniquely identified within the scope of the surrounding item element by its XML tag, which is why options (and array elements) do not require a *sid* attribute. In XFDL, there are two kinds of array elements: unnamed and named. An unnamed array element is surrounded by <ae> and </ae> tags. A named array element has its XML tag as its *sid*. Named array elements cannot begin with the tag <ae>. Further, since the XML tag of a named array element is a *sid*, the XML tag of a named array element must be unique within its parent element.

2.7 Document Reproducibility

XFDL processors are expected to preserve the XML prolog and epilog, the comments within the XFDL element, and all element attributes appearing in start tags but not specifically defined by XFDL. The

attributes must be associated with their respective start tags, and the comments must be associated with the respective pages, items, options, or array elements to which they apply. The XFDL processor must be able to reproduce these language components for signatures and for saving or transmitting the form.

3. The XFDL Compute System

An XFDL compute is an expression that controls the character content of an element at or below the element depth of an XFDL option. XFDL computes can be defined for custom options that are not in the XFDL namespace (a common practice used for computing intermediate results), but in such cases, the attribute must be properly namespace qualified (e.g. use xfdl:compute where the namespace prefix 'xfdl' is associated with the XFDL namespace URI). The XFDL compute expression appears in a *compute* attribute. This section defines the infix notation for XFDL compute expressions.

Most XFDL processors only need to preserve the compute as character data, but some applications must parse the text of computes and construct expression tree data structures to represent all computes in a form. This is necessary if the application must change the content of options or suboptions that are referred to by a compute. This section describes the syntax and operation of computes.

3.1 Whitespace in Computes

XFDL computes automatically support the notion of free form text found in most programming languages. With the exception of the contents of quoted strings (see section "3.5 Quoted Strings"), unlimited whitespace is permitted. Adding S? before and after every lexical token in every BNF rule in this section would unnecessarily obfuscate the presentation of what is essentially the standard BNF for mathematical and conditional expressions. Therefore, it is stated once here for the reader that all whitespace appearing outside of quoted strings and other lexical tokens is ignored.

While whitespace is formally ignored, there do arise cases in which it is necessary to use whitespace to properly communicate the expression. These situations arise because lexical analyzers are 'greedy' in the sense that they will match as much of a substring to the current token as possible. For example, in the boolean test a==b and c==d, the spaces before and after the keyword 'and' are required because the element reference token that matches 'a', 'b', 'c' and 'd' also matches 'band', 'andc' and 'bandc'.

The XFDL compute is a normal XML attribute and is therefore subject to all of the normal XML processing rules associated with attributes of the default type (CDATA). This includes whitespace normalization as well as entity and character reference resolution. An XML parser is expected to convert each tab or newline character into a space, so these characters should be avoided when text editing. XML processors capable of serialization are expected to encode tabs and new lines in attribute values as character references since the attribute value normalization on input implies that these characters would not be in the attribute value if they were not encoded as character references in the input. In raw XML text, a new line is encoded as
 which a subsequent XML parser will decode into a new line character in the attribute value.

3.1.1 Multiline Computes

Due to XML attribute value normalization, linefeeds are converted to spaces. Therefore, normal XML processing turns a multiline compute that has been carefully laid out by its author into a single line

compute. To alleviate this problem, form authors are encouraged to put the character reference
 at the end of each line of a multiline compute attribute.

A normal XML processor that reads
 followed by a new-line character will translate the value to a real new-line followed by a space. The serialization algorithm of an XML processor will then output the reference
 for the real new-line followed by a space. A subsequent parse by an XML processor will produce a real new-line followed by a space. Therefore, it is recommended that XFDL processors (which uses a normal XML processor for parsing input) serialize a real new-line followed by a space as the reference
 followed by a new-line, which this restores the multi-line appearance of computes. This behavior must not be performed when serializing the XML for a digital signature.

3.2 Structure of Mathematical and Conditional Expressions

An XFDL compute can be either a mathematical or conditional expression. A conditional expression has three parts separated by the ternary ?: operator. The first part is a Decision, which yields a boolean result. The consequences for a true and false boolean result recurse to the definition of Compute, permitting arbitrary nesting of decision logic.

[25] Compute ::= Expr | Decision ‘?’ Compute ‘:’ Compute

The decision logic can apply logical-or (|| or ‘or’), logical-and (&& or ‘and’), and logical negation (!) to the results of logical comparisons. The logical operators are left associative, and the comparison operators cannot be chained (e.g. a < b < c is illegal). The order of operations gives greatest precedence to negation, then logical-and, and least precedence to logical-or. To override this, parentheses can be used (e.g., the parentheses in (a<b || c<d) && e!=f cause the logical-or to occur first, and no parentheses are required if the logical-and should be performed first).

[26] Decision ::= Decision (‘||’ | ‘or’) AndDecision | AndDecision

[27] AndDecision ::= AndDecision (‘&&’ | ‘and’) NotDecision | NotDecision

[28] NotDecision ::= ‘!’ Comparison | Comparison

[29] Comparison ::= ‘(’ Decision ‘)’ | Expr (‘<’ | ‘>’ | ‘<=’ | ‘>=’ | ‘==’ | ‘!=’) Expr

A mathematical expression, denoted Expr, can include addition, subtraction, string concatenation (+.), multiplication, division, integer modulus, unary minus, and exponentiation. All mathematical operators are left associative except unary minus and exponentiation. Further, proper order of operations is

observed. Parentheses can be used to override the order of operations as shown in the non-terminal symbol named Value (defined later).

[30] Expr ::= Expr ‘+’ Term | Expr ‘-’ Term | Expr ‘+.’ Term | Term

[31] Term ::= Term ‘*’ NFactor | Term ‘/’ NFactor | Term ‘%’ NFactor | NFactor

[32] Nfactor ::= Factor | ‘-’ Factor

[33] Factor ::= Value ‘^’ NFactor | Value

3.3 Table of Operators

The following table details the operators permitted in XFDL:

Type of Operator	Symbol	Operation
Additive	+	addition
	- (minus)	subtraction
	+.	concatenation
Multiplicative	*	multiplication
	/	division
	%	modulus (returns remainder)
Exponentiation	^	exponential
Relational	>	greater than
	<	less than
	<=	less than or equal to
	>=	greater than or equal to
	==	equal to
	!=	not equal to

Type of Operator	Symbol	Operation
Logical	&& and || or !	AND AND OR OR NOT
Unary Minus	- (minus)	take negative
Decision	x?y:z	assign the value of expression **y** to the result if expression **x** evaluates to true. Otherwise, assign the value of expression **z** to the result.
Assignment	=	assign right operand to left operand
Membership	. (dot) [ ] ->	structure membership array membership indirect membership

3.3.1 Precedence of Operations

Operations are evaluated in the following order:

- membership
- exponentiation
- multiplicative and unary minus
- additive
- relational
- logical NOT
- logical AND
- logical OR
- conditional

3.3.2 Decision Operations and Namespace Qualification

Both decision operations and namespace use the colon (:) character as a token. In some cases, this makes it difficult to determine which colon is the decision operator token and which colon is the namespace token. For example, consider the following expression:

```
x ? custom:y : z
```

This is a variation on the common “if x then y, else z” expression. In this expression, the y statement is intended to be in the “custom” namespace. However, as it is currently written the expression does not clearly define which colon represents namespace and which colon represents the end of the y statement.

In cases such as these, parentheses should enclose the y statement of the expression, thereby providing a clear indication of where the y statement ends. For example:

```
x ? (custom:y) : z
```

3.3.3 Illegal Characters in XML Attributes

Note that the ampersand (&) and less-than (<) characters are not permitted in XML attribute values. Since XFDL computes appear in a compute attribute, these must be escaped with character or entity references (e.g. the entity references & for the ampersand and < for the less-than character). Hence, the less-than-or-equal symbol (<=) could be encoded as ‘ <=’.

3.4 Definition of Value

A value can be any of:

- a compute in parentheses, which provides an override for the order of operations.
- a quoted string (see section “3.5 Quoted Strings”).
- an XFDL reference to an element whose text data should be obtained when the compute is evaluated (see section “3.6 XFDL References to Elements”).
- the result of a function call (Section 3.8 Function Call Syntax).

[34] Value ::= ‘(’ Compute ‘)’ | qstring | XFDLReference | FunctionCall

3.5 Quoted Strings

A quoted string is used to express a literal value in XFDL. The language rules for computes permit the recognition of a quoted string token using the italicized token name *qstring*. Whitespace before the open quote and after the close quote is ignored. Because XML allows attribute values to be either singly or doubly quoted, the XFDL compute expression syntax allows both single and double quotes so that the XFDL author can avoid the use of entity references. Within an XFDL quoted string, any character is allowed except for the type of quote mark used to start the quoted string. Quoted strings can also be of arbitrary length in XFDL. To increase human readability, XFDL supports multiline string continuation. If the next non-whitespace character appearing after a closing quote is an open quote, then the closing quote, whitespace, and open quote are discarded from the input stream. .

```
[35] qstring ::= ((‘"’ [^"]* ‘"’) | (‘’’ [^’]* ‘’’))+
```

A literal must be quoted regardless of its type (i.e. character strings, numeric values, dates and so forth must all be quoted). However, the quoted string is in the XFDL compute attribute value, which is a quoted string in XML. Therefore, if double quotes are used to surround the compute expression, then literals must either be expressed with single quotes or by using character or entity references for double quotes (e.g. the entity reference "). Likewise, if the compute attribute is surrounded by single quotes, then either double quotes must be used for the quoted strings or character or entity references (such as ') must be used to express quoted strings.

Occasionally a literal value simply must include the type of quote mark used to surround the literal. In such cases, the XFDL author can either use entity references to surround the literal value or use backslash (\) escaping. For example, \’ can be used to place a single quote in an XFDL quoted string that is surrounded by single quotes.

Backslash escaping is also supported for several other characters. The escape sequences \n and \t result in a new line and a tab, respectively, in the quoted string content. Since the backslash is the escaping character, it must also be escaped to be inserted into the string content (e.g., \\).

Note that since a quoted string is meant to be interpreted by the XFDL compute expression parser rather than the XML processor, it is recommended that backslash escaping be used rather than XML character and entity references. However, it is safe to use the XML mechanisms except for encoding a new-line in a quoted string. The special processing performed by XFDL processors to preserve the text layout of multiline computes may conflict with the use of an XML character reference for a new-line if it is followed by a space (please see “3.1.1 Multiline Computes”). While correct processing will still result due to attribute value normalization on the next parse, the text layout and surface string of the XFDL will be changed.

3.6 XFDL References to Elements

The simple character content of options and suboptions are obtained as the operands of XFDL compute expressions using XFDL references. XFDL references support forward and backward referencing. An XFDL reference can refer to any option or array element with simple character content.

The element containing the desired character content is identified using scope identifiers to negotiate a path through the parse tree. To traverse through the page and item levels and identify an option, the well-known ‘dot’ membership operator is used. The well-known square-bracket array notation can then be used to access suboptions to arbitrary element depth. For example, *Page2.Field2.value* would access the option with tag name <value> in the <field> element having a *sid* of **Field2** in the <page> having a *sid* of **Page2**.

Because each XFDL element’s scope identifier (*sid*) is used to uniquely identify an element only within the surrounding parent element, XFDL can support relative referencing. For example, in an element identified as Field1, if a computation includes the reference *Field2.value*, this means that the character data of the value option in the item Field2 *on the same page* will be obtained. If Field2 is on a separate page, say Page2, then a compute in Field1 can still access its value using the fully-qualified reference *Page2.Field2.value*.

The context for interpreting a reference is also decided based on the form of the reference itself in combination with its location in the form. For example, in the XFDL below, the reference *Bill.value* appears in a compute that is attached to a grand-child suboption of the format option. However, the name after the rightmost ‘dot’ operator always refers to an option, and the name before the rightmost dot always refers to an item. Since the page is not specified, it is determined to be the page containing the reference.

```
<page sid= "CreditCardApp">

   <field sid="Bill">
      <label>Your monthly bill is:</label>
      <value>700</value>
      <editstate>readonly</editstate>
   </field>

   <field sid="MinPayment">
      <label>Enter payment amount:</label>
      <value></value>
      <format>
         <ae>dollar</ae>
         <range>
            <ae compute="Bill.value * ’0.05’">35</ae>
            <ae compute="Bill.value">700</ae>
         </range>
      </format>
   </field>

</page>
```

XFDL references can also grow arbitrarily below the option level using the array notation, allowing access to unbounded array element depth within any option. If an array element is not named, then the zero-based numeric position of the array element is used in the square brackets. If the array element is named, then the scope identifier can be used in the square brackets. For example, given the format option of the XFDL above, the reference *format[0]* yields **dollar** and the reference *format[range][1]* yields **700**. If a suboption is named, the numeric position can still be used, e.g. *format[1][1]* also yields **700**.

The above description covers static references. The XFDL referencing model also supports dynamic references. The left associative ‘arrow’ operator (->), also known as the indirect membership operator, expects to receive a static or dynamic reference as a left operand. The run-time value of the static or dynamic reference must conform to the syntax of the ItemRef non-terminal. The right operand of the indirect membership operator is an option reference. At run-time, the left operand is evaluated, yielding a static item reference to an XML element representing an XFDL item. This run-time item reference is combined with the right operand of the indirect membership operator to yield an option or array element whose simple data is the result of the evaluation.

The simplest example of a dynamic reference is retrieving the text of the selected cell in an XFDL list box or popup, as is discussed in section “7. Details on Items”, because the *value* option of a list or popup is equal to the item reference of the selected cell item. Thus, given an example popup that offers a selection of days of the week, the text for the day of week selected by the user is obtained by *Popup_DayOfWeek.value->value*.

An option reference can simply refer to element tag names in the XFDL namespace without any namespace qualification. A namespace qualified scope identifier (the non-terminal NSsid below) must be used if the element being referenced is not in the XFDL namespace. In order to reference element in the empty namespace, XFDL supports the predeclared prefix *null*. For example, to reference an element *E* which has an empty namespace URI, use the reference *null:E*.

Below are the syntax rules for an XFDL reference. Note that unlike most other syntax rules for XFDL expressions, intervening whitespace is not allowed in an XFDL reference. An XFDL reference is treated as a single lexical token.

```
[36] XFDLReference ::= StaticRef | StaticRef ‘->’ DynamicRef
[37] StaticRef ::= ItemRef ‘.’ OptionRef | OptionRef
[38] ItemRef ::= ((sid ‘.’)? sid ‘.’)? sid
[39] DynamicRef ::= DynamicRef ‘->’ OptionRef | OptionRef
[40] OptionRef ::= NSsid (‘[’ (Digit+ | NSsid) ‘]’)*
[41] NSsid ::= sid | (Letter (Letter | Digit | ‘_’)*) ‘:’ sid
```

3.6.1 Referencing the XFDL Version

Since the XFDL version is represented by the XFDL namespace URI declaration in the root XFDL node, the normal *page.item.option* notation cannot reference the version number directly. However, as a convenience, it is allowable to refer the XFDL version as a global form option, as follows:

```
global.global.version
```

XFDL processors are expected to recognize this notation.

3.7 Function Call Syntax

Function calls run code that may be external to the XFDL form definition. A set of predefined functions (called *system* functions) for doing standard mathematical operations, string manipulations, and so on, is given in section “9. Function Calls”. The LibName allows functions to be grouped into separate namespaces, but the predefined system functions do not require a LibName. The names of the system functions are considered reserved words and should not be used as function names in other function libraries.

[42] FunctionCall := (LibName ‘.’)? FunctionName ‘(‘ (Compute (‘,’ Compute)*)? ‘)’

[43] LibName ::= sid

[44] FunctionName ::= sid

3.8 Representing and Running XFDL Computes

3.8.1 Introduction

The XFDL compute engine implements a ‘declarative’ computation system. The behavior of the algorithm is similar to that of a spreadsheet. Complicating factors for XFDL include dynamic references and side effect functions such as *set()*. When a form is first started, many nodes with computed values may need to be evaluated to find current display values. After the form is started, any specific change to a given node should result in updates to computationally dependent nodes. Further, the process is recursive in that when a computationally dependent node is updated, then it may have further computationally dependent nodes that must be queued for update.

The XFDL specification provides an abstract version of the desired algorithm, but does not place specific constraints on the data structure used to represent the compute system.

3.8.2 Cached Dependency Lists

On form startup, the compute system associates with each form node *F* a list of other form nodes that are computationally dependent on *F.* A node is dependent on *F* if a reference to *F* appears in the node's compute. Figure 1 expresses the XFDL compute engine algorithm based on utilizing these dependency lists. The dependency lists are viewed as a *directed graph*, or digraph, of computational dependencies.

Each form node will have several flags associated with it: Visited, OnStack, Processed, Processing, and UsedToDeref. The **Visited** flag is require by depth first search. The **OnStack** flag prevents duplicate entries on the stack, which prevents circular referencing. The **Processed** flag is used in conjunction with OnStack to allow a form node to stay on the stack but not be processed again when the form node's dependencies have all been processed. The **Processing** flag is used when *XFDLRunComputes()* is called recursively from within the *eval()* function (side effect functions such as *set()* call *UFLSetLiteral()*, so the compute system is called recursively during their evaluation). The Processing flag tells *XFDLRunComputes()* to terminate, returning control to *eval()*. The **UsedToDeref** flag indicates whether a given form node's literal value is being used by any compute in the form to dereference another form node (explained in the next section). If this flag is set, then changing the node's literal will cause a change to the dependencies in the form.

The dependency lists referred to in Figure 1 are constructed at build time. For each compute C_I, the function *CreateRefList()* traverses the compute parse tree for all references to existing nodes. For each reference in C_I to a form node F_R, the form node F_P (the parent node containing C_I) is added to the dependency list of F_R, except when F_P is equal to F_R (this exception allows support for self-referential computes as discussed below). Thus, when a node F_R changes, the new algorithm has a prebuilt list of dependent computes that must be re-evaluated.

The outermost loop of the new algorithm runs until it reaches quiescent state (in other words, it achieves closure on the change to element E). When E is null, the new algorithm pushes all computes for reevaluation. The inner loops of the original algorithm in Figure 1 were mainly designed to find which computes were pertinent (in other words, which ones needed to be re-evaluated). In the new algorithm, the new pertinent vertices of a changed vertex F_P are immediately known due to the precomputation of the dependency lists.

The implied behavior of the outermost loop is that it should terminate if there is a circular reference other than a self-reference. By omitting a node from its own dependency list, self-referential computes do not become circular references. However, circular references involving more than one node can be expressed using static references (for example, A=B, B=C, C=A), dynamic references, or even indirect referencing via XFDL function calls (for example, the assignment X = A+set("A", A+ "1")). To prevent circular referencing, the algorithm uses a ChangeStack rather than a ChangeList (queue), and it uses the

flags OnStack and Processed to implement a *depth first search*, which is an algorithm that finds cycles in a search space. *Push()* will not push a node already on the stack.

Figure 1: Algorithm Sketch for XFDL Compute Engine

```
RunXFDLComputes(F (form), C (computes), E (a changed element or null)) ::=
If E is not null, then ProcessLiteralChange(F, E)
Else   For each c ∈ C, ParentFormNode(c).Visited = 0
          For each c ∈ C,
             If not ParentFormNode(c).Visited,
                DFSPush(F, ParentFormNode(c))
       For each c ∈ C, ParentFormNode(c).Visited = 0

While not Empty(F.ChangeStack) and not TopStack(F.ChangeStack).Processing
Do
   FP = TopStack(F.ChangeStack)
   If FP.Processed, then Pop(F.ChangeStack);
   else   FP.Processing = on
          Literal = eval(ChildComputeNode(FP), F.ChangeStack)
          If cval(FP) ≠ Literal, then
             cval(FP) = Literal
             ProcessLiteralChange(F, FP)
          FP.Processing = off
          FP.Processed = on

ProcessLiteralChange(F, E) ::=
If E.UsedToDeref, then
   ProcessDependencyChanges(F, E)
DFSPush(F, E)
For each entry e of F.ChangeStack from top to bottom,
   If e.Visited, then e.Visited = 0
   Else break loop
F.TotalChangeList = F.TotalChangeList ∪ E

DFSPush(F, E) ::=
E.visited = 1
For each f ∈ DependencyList(E)
   If not f.Visited and not f.OnStack
      DFSPushComputes(F, f)
Push(F.ChangeStack, E)
```

```
ProcessDependencyChanges(F, E) ::=
For each f ∈ DerefSubset(DependencyList(E)),
   NewRefList = CreateRefList(f, F)
   For each r ∈ (f.RefList ∪ NewRefList) - (f.RefList ∪ NewRefList)
      If r ∈ f.RefList, then
         DependencyList(r) = DependencyList(r) - f
      Else
         DependencyList(r) = DependencyList(r) + f
   f.RefList = NewRefList

Pop(S) ::= f = S.Pop(); f.OnStack = f.Processed = off

Push(S, f) ::= if not f.OnStack, then
      S.Push(f);
      f.OnStack=on;
      f.Processed=off
```

Note, however, that we do not generate an error when a duplicate is found on the change stack. This is because the loop initialization pushes all computes when E is null. Thus, there are valid cases where we want to ignore the duplication without generating an error. Reporting true circular references can be done using a depth first search at design time with the compute system turned off.

3.8.3 Topological Sorting

When a node E changes, the function *ProcessLiteralChange()* does not simply push the elements in its dependency list, which would make **XFDLRunComputes** a simple depth first search of a digraph. Instead, it calls *DFSPush()*, which explores the computational dependencies of E using a depth first search. In other words, E is treated as the root of a depth first search tree of computational dependencies. *DFSPush()* does not push E until it has visited all of E's descendants (post-order visitation of the DFS tree). This has the effect of placing the dependencies in a linear order on the change stack (linear ordering in a directed acyclic graph is called topological sorting). Since an element E appears on the stack above its descendants, it is re-evaluated before its descendants. Since E's descendants are dependent on E, evaluating E before its descendants ensures that E will have the correct value before its descendants are evaluated.

Because the dependency graph is directed, it is still possible for a depth first search to take exponential time because it must explore a subtree rooted at *r* for each parent of *r*. The directed graph can be acyclic even though the corresponding undirected graph has cycles. If those cycles have a repeated structure (such as might be found in a form with many rows of identical construction), the work on each row can be constant but the work of a row may occur once for each change to the preceding row. If each row requires at least two units of work, then the computation is exponential in the number of rows.

3.8.4 Handling Dynamic References

A dynamic reference is a compute that includes the use of the arrow operator to obtain a value by dereferencing other values in the form. Dynamic references imply changes to the dependency lists during run-time. A number of enhancements are required to solve this problem efficiently.

The dependency list of each form node will be segregated into normal dependencies and dereference dependencies. The form node's UsedToDeref flag will be set if and only if its dependency list contains dereference dependencies. After changing the literal value of a form node whose UsedToDeref flag is set, certain dependencies will need to be re-evaluated (described below).

For each form node f containing a compute, we store a ReferenceList containing all form nodes referred to within the compute. Again, the list will be segregated into normal and dereference references. All subreferences in a dynamic reference except for the rightmost reference are classified as dereference dependencies. For example, in a compute containing *popup.value->value*, if the popup's value is "cell1", then the literal of *popup.value* is dereferenced, but *cell1.value* is not. This distinction is important because a change to *popup.value* will cause dependency changes whereas changing *cell1.value* will not.

Actually, the reference list of each compute must already be built as part of setting up the dependency lists. The reference lists are required since, for each form node F_P in f.ReferenceList, we must add *f* to F_P.DependencyList. Now we are simply deciding to retain the list for use in solving problems introduced by dynamic references. So, while building dependency lists, if F_P is classified as a dereference in *f*.ReferenceList, then *f* will be classified added a dereference dependency in F_P.DependencyList.

The *ProcessDependencyChanges()* function mentioned in Figure 1 can then rebuild the dependency lists efficiently. When processing a stack entry F_P, if the UsedToDeref flag is set, then some dependency lists may need to be changed. A form node *f* is considered to be in the 'dereference' portion of F_P.DependencyList. Begin by creating a new reference list for *f*. Any form node F_P in *f*.ReferenceList but not in the new reference list implies the removal of the dependency entry *f* from F_P.DependencyList. Any form node F_P not in *f*.ReferenceList but in the new reference list implies the addition of the dependency entry *f* to F_P.DependencyList. Finally, the new reference list is assigned to *f*.ReferenceList.

Note that most of the dependency changes involve the addition and deletion of 'normal' references and dependencies. For example, given a form node *f* with a compute of *popup.value->value*, if *popup.value* were to change from "cell1" to "cell2", then the 'normal' entry of "cell1" in the *f*.ReferenceList would be deleted, and a 'normal' entry of "cell2" would be added. Furthermore, the normal dependency on *f* would be removed from in *cell1.DependencyList* and added to *cell2.DependencyList*. It is possible that reference list and dependency list entries of type 'dereference' can also be modified. An example would be a double dereference; in other words, the change of *x.value* in a form containing the compute *x.value->value->value*. Thus, an entry *f* in F_P.DependencyList is a dereference if changing node F_P implies the need to rebuild *f*.ReferenceList (which in turn implies changes to other dependency lists, possibly including F_P.DependencyList).

The statements above assert that the algorithm will only rebuild the references lists of nodes marked as ‘dereference’ dependencies in F_P.DependencyList. Due to the high cost of calling *UFLDereference()*, the algorithm should not rebuild reference list entries that result from static references. Static reference nodes are distinguished from dynamic reference nodes in the compute tree. Note that the leftmost subreference of a dynamic reference is also static, so it will also not be recomputed. Instead, the value in the reference cache will be used (see section “3.8.5 Reference Caching”).

3.8.5 Reference Caching

When a compute is parsed into a parse tree of compute nodes, the nodes representing static references currently cache the results of *UFLDereference()* so that future evaluations of the compute can proceed without a costly search of the form. Although the ReferenceList of each node contains all references, these cached references are stored in compute nodes for instantaneous access by *eval()*.

In the current API, caching of dynamic references was not performed because the implementation had no way of knowing whether a change occurred that would affect the validity of the cached value. The algorithm is able to cache dynamic references. When the ReferenceList of a given form node is reconstructed, the dynamic references in the compute node associated with that form node will also be re-evaluated and re-cached.

Note that dynamic references call *UFLDereference()* once for the leftmost subreference plus once per arrow in the dynamic reference. The reference cache will cache the last *UFLDereference()* as this is the final results required by the *eval()* function. However, the first *UFLDereference()* on the leftmost subreference will also be cached. The leftmost subreference is static, so storing it can be used to optimize the process of resolving dependency changes (see section “3.8.4 Handling Dynamic References”).

3.8.6 Re-entrancy

XFDLRunComputes() is no longer shown as returning a value. This is because the returned change list, formerly denoted Z, is actually a property of the form F and is now denoted F.TotalChangeList. This member is initialized to emptiness when the form is first created. It is updated by running computes, but it is only reinitialized to emptiness when an application takes ownership of the list from the form. This allows an application to make numerous changes to the form before using the F.TotalChangeList to update structures external to the form (such as a database or a corresponding GUI).

The change list itself is now also a property of the form, denoted F.ChangeStack, and it also is initialized to emptiness during form creation. Notice that it is not initialized to emptiness at the start of run computes. This is done to support re-entrancy. If the call to *eval()* runs a side effect function that in turn calls *UFLSetLiteral()*, then *XFDLRunComputes()* will be reinvoked. The new function instance will continue attempting to process the same change stack. Since the change stack forbids duplicates, *XFDLRunComputes()* cannot recurse indefinitely on a form of bounded size (side-effect functions such as *duplicate()* can create new pieces of form with each run, so if the *duplicate()* duplicates itself, an infinite loop can occur). When the instance of *XFDLRunComputes()* that was called recursively

encounters a stack entry marked ‘Processing’, this indicates that it is time to return back to *eval()*, which ultimately returns to the previous instance of *XFDLRunComputes()*.

3.8.7 Duplicate Entries on the Total Change List

The algorithm in Figure 1explicitly shows how it uses the OnStack and Processed flags of each form node to prevent duplicate entries in the change stack. The algorithm also uses the set union operator to indicate that duplicates will not be allowed on the TotalChangeList. This is done using another flag, OnChangeList, which is cleared on form node creation. An attempt to add a form node to the TotalChangeList is preceded by a test of this flag. If it is set, then the form node is already on the change list and will not be added again. If the flag is clear, then the form node will be added to the TotalChangeList, then its OnChangeList flag will be set.

When the API function *UFLGetChangeList()* is called, the OnChangeList flags of each element in the TotalChangeList must be cleared before passing ownership of the TotalChangeList from the form to the API caller.

3.8.8 Missing References

If an XFDL reference refers to a non-existent element or an element with array content, then the reference simply resolves to the empty string. If a dynamic subreference (i.e. the left operand of the dereference operator) refers to a non-existent element, then the containing compute is disabled. If the desired form node is created later, the compute is immediately re-activated.

3.8.9 Handling of Element Deletion

When an element is deleted, it may contain a compute. The XFDL processor must remove dependency list entries associated with the elements referenced by the compute expression being destroyed. Moreover, references to the element being deleted may appear on change lists and reference caches in the compute system. These must be removed as well. If a compute refers to an element that is being deleted, then the compute will be handled according to the rules for missing references (see the previous section for how these are handled).

3.8.10 Limitations

It is possible to create cases in which the algorithm halts a sequence of computations that do not technically form circular logic, especially when side effect functions such as *set()* are used. In other words, the case would halt by itself if the algorithm would allow duplicates on the change stack. However, there is a difference between circular logic and circular references. According to the XFDL specification, circular referencing is forbidden (except for self references), and the compute system simply does not continue evaluation of a node C that is already on the stack.

A second limitation, also involving side effect functions, can actually cause an infinite loop in the algorithm. The problem occurs when a side effect function creates a new portion of the form. If the new portion of the form contains computations that cause the continued recursive creation of new portions of the form, then each new portion of form has elements that are distinct from all previous elements and hence do not technically even cause a circular reference. A depth first search halts on any form of finite size, but if computations result in unbounded form growth, then it is correct behavior for a depth first search to run indefinitely exploring the new regions of the form.

A third limitation is that a given compute may still be fired more than once. This is not the result of circular reference in the dependency digraph but rather that the same descendant is reachable along multiple paths (the undirected graph corresponding to the dependency digraph has cycles, but the digraph contains no way to get from the descendant back to the ancestor). In other words, a node *r* has multiple parents because a depth first search of a digraph can yield multiple DFS trees each containing *r* with one of its parents.

To solve this problem with static references only, a linear ordering could be created for the directed acyclic graph of dependencies, and the edges of each node could be resorted (in linear total time) so that the push order would respect the linear ordering. However, this approach is impractical for XFDL. Due to dynamic referencing, the linear ordering would need to be maintained dynamically. Furthermore, dynamic changes to the linear ordering would cause dynamic changes to the order of elements in the change stack. More advanced methods could be developed to account for this, but the problem expands when the implicit dependencies introduced by side effect functions are taken into account. Although the compute system could run faster by running less computes, the cost of operations necessary to account for dynamic references is prohibitive, and implementing the method would not be worth the trouble due to the inability to achieve correctness with side effect functions. Thus, the linear ordering imposed by *DFSPush()* is temporal, and changes to nodes used in dynamic references may invalidate the order such that extra computes will sometimes run more than once. Those who wish to prevent functions such as viewer.MessageBox from running more than once must still resort to protecting these computes with conditional logic and the *toggle()* function.

4. Signatures in XFDL

As discussed previously (in section “1.6 Non-Repudiation and the Document-Centric Model”), digital signature technology provides part of the solution for creating non-repudiation in a document-centric model. As such, XFDL supports the application of digital signatures to any XFDL document. However, to properly implement support for digital signatures, both common use scenarios and overall security must be considered.

For example, signatures are often used to sign only a portion of a document. Furthermore, a secondary signature is often used to sign the rest of the document while also endorsing the first part of the document. The classic example of this is the “For Office Use Only” section in any form. The implementation of digital signatures in XFDL must support scenarios like this, allowing both for filtering of what is signed and for overlapping signatures.

Furthermore, while digital signatures clearly identify the user, the application of digital signatures must also add a measure of security to the document itself. That is, once a document is signed, it should be impossible to change any of the information that was signed. Thus, a number of algorithms and rules must be enforced by the XFDL processor in use.

4.1 Applying Signature Filters

XFDL supports a filtering system for signatures. In effect, this allows any combination of form elements to be either included or excluded from a signature, which in turn allows forms to be divided into logical sections for the purposes of signing.

For example, a document may include a “For Office Use Only” section that should not be signed by the original user. By applying the correct filters, this section can be excluded from the signature, allowing office workers to complete those portions of the document even after the document has been signed.

XFDL includes a series of signature filters. Each filter applies to a different cross-section of XFDL elements. For example, the *signitems* and *signitemrefs* filters control which items are signed or ignored, while the *signoptions* and *signoptionrefs* filters control which options are signed or ignored. Each level of filter also has an assigned order of precedence. For example, filters at the option level override filters at the item level.

By using these filters in combination, XFDL provides complete control over which elements are omitted from a signature (or alternately to indicate which elements should be included in a signature, though ‘inclusive logic’ filters should be used sparingly and with great care).

The complete list of available filters is: *signitems*, *signoptions*, *signpagerefs*, *signdatagroups*, *signgroups*, *signitemrefs*, and *signoptionrefs*. These filters are implemented as XFDL options, and are detailed later in this document. The order of precedence for these filters is outlined in “Appendix C: Order of Precedence of Filters” on page 381.

4.2 Namespaces in Signature Filters

Some signature filters refer to the elements to be omitted (or included) by their element tag names. These filters are *signitems*, *signoptions* and *signoptionrefs*. These options are lists whose members are compared to element tags. The comparison is namespace aware. For example, if *<ae>xfdl:field</ae>* is a member of the *signitems* filter, then the member will match any item-level element with the local name *field* and a namespace URI equal to the one bound to the prefix *xfdl*. Note that if a namespace prefix used in an element tag is not mapped to the same namespace URI as it is in a signature filter member, then the signature filter member will not match the element. For example, if the xfdl prefix is mapped to the XFDL namespace URI in the above signitems member, then the following elements will match:

```
<xfdl:field xmlns:xfdl="http://www.PureEdge.com/XFDL/6.0">
<field xmlns="http://www.PureEdge.com/XFDL/6.0">
<foobar:field xmlns:foobar="http://www.PureEdge.com/XFDL/6.0">
```

but it will not match:

```
<xfdl:field xmlns:xfdl="http://foo.bar">
<foobar:field xmlns:foobar="http://foo.bar">
<field xmlns="">
```

The last <field> element above has an empty namespace URI. To match such an element, the *null* prefix must be used, e.g. *<ae>null:field</ae>*.

The namespace prefixes used in evaluating each member of a signature filter are obtained from the namespace context of the element containing the signature filter member. Each member of a signature filter may define different namespace prefixes. However, the default namespace is the XFDL namespace. For example, the member `<ae>field</ae>` in *signitems* matches any item-level <field> element with a default namespace URI equal to the XFDL namespace URI as well as namespace qualified elements such as <xfdl:field> if the given prefix is bound to the XFDL namespace URI.

Signature filter members that are namespace qualified with a prefix bound to the XFDL namespace URI are logically equivalent to signature filter members that are associated with the XFDL namespace URI by the default namespace. For example, a *signitems* member of <ae>field</ae> with the default namespace of XFDL is logically equivalent to the following, where the prefix *xfdl* is bound to the XFDL namespace URI:

```
<xfdl:ae>field</xfdl:ae>
<xfdl:ae xmlns="http://foo.bar">field</xfdl:ae>
<xfdl:ae xmlns="">field</xfdl:ae>
```

Suboptions of a signature filter that are not in the XFDL namespace are ignored. For example, the *signitems* suboption *<ae xmlns="">field</ae>* has no effect on signature filtration.

4.3 Applying Multiple Signatures

Documents often require multiple signatures. Furthermore, it is common practice for some signatures to endorse other signatures. These secondary signatures can be said to overlap the original signatures, since they sign both the document and the original signature.

For example, an insurance claim requires the claimant to sign the document. Later, the insurance adjuster may also have to sign the document, both to endorse the information provided by the claimant and to endorse information they have added to the claim.

XFDL allows any number of signatures in a single document. The signatures will sign separate portions of the form, or will overlap with other signatures, as specified by the filters used.

For example, the first signature may use a set of filters that includes all elements in the top half of a page. The second signature may use a filter that includes the first signature and the top half of the page. Finally, a third signature might use a filter that includes the entire page and both the first and second signatures.

4.4 Securing Signed Elements

Paper documents rely on ink to secure the document. That is, once a document is signed, it is difficult to change the document because it is difficult to erase ink from paper. The very nature of paper and ink enforce the security of the document, since attempts to change the document generally leave detectable traces.

In contrast, digital documents do not share this type of inherent security. In fact, most digital documents allow easy and undetectable modification. For example, word processing documents are easily opened, changed, and saved again without leaving any evidence of what, if anything, was changed or who made the changes.

For this reason, the XFDL processor must provide the necessary security. Once an element in a document is signed, it is implicit that future readers should be unable to change that element. Thus, once an XFDL document is signed, the XFDL processor must ensure that those elements included in the signature filters cannot be changed.

This feature is not explicitly supported by XFDL, in that there are no flags in the language that indicate whether a particular element in a document is signed. Instead, the software processing the XFDL must interpret existing signatures and enforce the rules correctly based on the filters in those signatures.

4.5 Preventing Layout Changes

Once a document is signed, it is also implicit that the layout of that document should be secure. For example, if it were possible to move a paragraph, or even a line, the meaning of the document could be changed.

To reflect this, any software processing XFDL must maintain the position of signed visual elements. This means that both the position and the size of the visual elements must be secured. If a visual element can change size, then it could be enlarged to obscure another visible element and thereby change the meaning of the document. Clearly, this must be prevented.

Thus, when a document is signed, the width, height, and position of all visible signed elements must be recorded. XFDL provides the *layoutinfo* option as a place to store this information within a given signature element. Furthermore, the *layoutinfo* option itself should be signed as part of the signature, ensuring that it cannot be changed.

The layout can later be tested by re-calculating the position of all signed elements and comparing this to the information stored in the layoutinfo option for that signature. If the information does not match, then the document has been modified and cannot be trusted.

The software processing the XFDL should perform this layout test at the following times:

- Immediately after a signature is created, it should test the entire document. This ensures that the process of generating the signature did not change the information.
- Whenever a page of the document is viewed, it should test the signed contents of that page.
- Whenever an item is computationally added, deleted, or moved, it should test the appropriate page.
- Whenever the details of a signature are viewed, it should test all portions of the document signed by that signature.

4.6 Preventing Exploitable Overlaps of Signed Elements

Unlike paper documents, digital documents also offer the potential for visual elements to overlap. For example, it is possible to create a block of text in a document, and to then obscure or hide that text with a second, overlapping block of text. In this scenario, even if the first block of text was secured with a signature, it would be possible to move the second block of text after the document was signed. This would change the meaning of the document by revealing information that was previously hidden.

Since the guiding principle of signatures is that “you sign what you see,” a scenario in which visual items are hidden or significantly overlapped cannot be allowed. If the signer cannot see elements of the form, then the signature cannot be considered valid.

When a document is signed, the XFDL processor must ensure that none of the signed visual elements are overlapping with unsigned visual elements. If an overlap is detected, the software must either warn the user or prevent the signature from being created. Furthermore, if an existing signature is found to include elements that are overlapping with unsigned elements, the document has been altered and the software must warn the user.

However, this test must allow for certain tolerances. Most of the visual elements in an XFDL document are surrounded by a small border of unused space which can be allowed to overlap without obscuring the item itself. For example, two labels might overlap slightly without the text in either label being obscured. In fact, this sort of overlap is often necessary when reproducing tightly spaced paper forms. Thus, an overlap of two pixels should be allowed for each item.

This test may also ignore signed elements that overlap each other, since the layout tests discussed earlier prevent signed elements from being moved. Furthermore, this test must also make exceptions for box items. Boxes are often used to visually create sections in a document, and will overlap other visual elements as a result. This overlap is allowed in the following cases:

- A signed box can overlap with any unsigned item, with the exception of other boxes.
- A signed box can overlap with an unsigned box if the unsigned box appears on top of the signed box (that is, if the unsigned box comes after the signed box in the XFDL serialization, such that it is drawn after the signed box). This allows the desired behavior of signing a large box but allowing unsigned items (including boxes) to appear on top of part of the signed box, and it disallows the unsigned box from later being moved in the XFDL serialization such that it disappears. Note that an unsigned box can be added after signing and allowed to overlap a signed box that was previously unobscured, but the other overlap rules prevent this from happening if the unsigned box overlaps other signed items in the signed box. To protect empty spaces in signed boxes from being obscured by unsigned boxes, the form author should place empty signed transparent labels in the spaces.

The XFDL processor should perform this overlap test at the following times:

- Immediately after a signature is created, it should test the entire document. This ensures that the process of generating the signature did not create overlaps.
- Whenever a page of the document is viewed, it should test that page.
- Whenever an item is computationally added, deleted, or moved, it should test the appropriate page.
- Whenever the details of a signature are viewed, it should test all portions of the document signed by that signature.

5. Small XFDL Form Examples

The first example in Figure 2 is designed to show a whole XFDL form. After the XML prolog, the root XFDL element declares the XFDL namespace URI to be the default namespace, and it binds the prefix *xfdl* to the XFDL namespace prefix. Implicit in the XFDL namespace URI is the XFDL language version (6.0) to which the form complies. There is a form global variable stating that all pages should have a medium gray background color given by the RGB triplet (128, 128,128). However, the page global background color is set to RGB (192, 192, 192) — light gray. Since page globals override form globals, the page will have a light gray background.

The background color option uses element depth to express an array for the RGB triplet. However, XFDL options are consistent in their use of element depth, so the background color would be specified using an <ae> suboption even if only one were needed to declare a color value by name (see “Appendix D: Color Table” for a list of valid color names in XFDL).

The page global item also contain a *label* option that declares the caption bar text for the window used to display the page. Note that *label* is a keyword that is used both as an item type and an option scope identifier. Widgets such as fields and comboboxes can have text labels associated with them, but image and text labels can also be placed anywhere on the form, so a separate label item is required in the language.

After the global options, the page contains three fields: the first two collect side lengths for a right triangle; the third computes the length of the hypotenuse of the right triangle with the given side lengths. An editstate of readonly is given to prevent the user from accidentally destroying the compute by entering a value for field C.

Figure 2: A Simple XFDL Form

```
<?xml version="1.0"?>
<XFDL xmlns="http://www.PureEdge.com/XFDL/6.0"
   xmlns:xfdl="http://www.PureEdge.com/XFDL/6.0">
   <globalpage sid="global">
      <global sid="global">
         <bgcolor>
            <ae>128</ae>
            <ae>128</ae>
            <ae>128</ae>
         </bgcolor>
      <global>
   </globalpage>
```

```
    <page sid="Pythagorean_Theorem">
        <global sid="global">
            <bgcolor>
                <ae>192</ae>
                <ae>192</ae>
                <ae>192</ae>
            </bgcolor>
        </global>
        <label>Pythagorean Theorem Form</label>

        <field sid="A">
            <label>Enter A:</label>
            <value>3</value>
        </field>
        <field sid="B">
            <label>Enter B:</label>
            <value>4</value>
        </field>
        <field sid="C">
            <label>Hypotenuse length C:</label>
            <editstate>readonly</editstate>
            <value compute="sqrt(A.value^’2’ + B.value^’2’)"
                >5</value>
        </field>
    </page>
</XFDL>
```

The second example in Figure 3 (below) omits the XML prolog and the declarations for the root XFDL element and page. The example only shows two items. It is designed to demonstrate deeper element depth and more computes than the form shown in Figure 2.

The first item is a field that purports to ask the user what portion of a bill, such as a credit card bill, will be paid. The *format* option contains a number of array elements. The first of them contains the word ‘dollar’ and represents the type of user input that will be permitted in the field. In the typical *format* option (see section “8.18 format”), the input type is not named and would therefore appear between the *<ae>* and *</ae>* tags. The second and third array elements in the *format* option are also unnamed. They provide additional information about the format, such as the fact that user input is mandatory (i.e., emptiness is not a permitted response), and that a dollar sign should be appended to the user’s input.

The last array element declared in the *format* option in Figure 3 is named ‘range’, and it contains an array of two elements that define the lower and upper bounds of the user’s input. For a credit card bill, the range of payment is typically bounded above by what the cardholder owes and bounded below by some small percentage of the current balance. Thus, the *format* option shows the possibility of

unlimited array element depth as well as the inclusion of computes deep within the element hierarchy. XFDL offers what is known as a fine-grain compute system.

The second item element in Figure 3 is a label that demonstrates a longer compute expression, including several array element references. Note that at the end of the compute, the 700 is concatenated to the end of the string rather than added to the 35. Because addition is left associative, the entire portion of the string prior to the 700 has already been constructed. Therefore, due to run-time type identification, the last + operator performs string concatenation, not a mathematical addition.

Figure 3: Example of Suboption Array Elements

```
<field sid="PayNow">
   <label>
      What portion of this bill do you want to pay now?
   </label>
   <value></value>
   <format>
      <type>dollar</type>
      <ae>add_ds</ae>
      <ae>mandatory</ae>
      <range>
         <ae compute="Balance.value * ’0.05’">35</ae>
         <ae compute="Balance.value">700</ae>
      </range>
   </format>
</field>
<label sid="DemonstrateSuboptionReferencing">
   <value compute="PayNow.format[type] + ’ ’ + &#xA;
      PayNow.format[1] + ’ ’ +PayNow.format[range][0] &#xA;
      + PayNow.format[range][1]"
      >dollar add_ds 35700</value>
</label>
```

6. Global Settings

6.1 Form Globals

Form globals specify particular settings for the form and determine its physical characteristics. For example, the *bgcolor* option determines the background color of all pages in the form. Form globals appear within the *global* item in the *globalpage*, which appears at the top of a form. Options defined within a page or item can override global settings for that particular page or item.

Available Options

activated, bgcolor, border, dirtyflag, focused, fontcolor, fontinfo, formid, keypress, label, next, previous, printbgcolor, printfontcolor, printsettings, saveformat, transmitformat, triggeritem, version

---

Note: *For descriptions, see section “8. Details on Options and Array Elements”.*

Example

This example defines settings and characteristics for the form:

```
<?xml version="1.0"?>
<XFDL xmlns="http://www.PureEdge.com/XFDL/6.0"
   xmlns:xfdl="http://www.PureEdge.com/XFDL/6.0">
   <globalpage sid="global">
      <global sid="global">
         <formid>
            <title>TimeSheet</title>
            <serialnumber>{D5D768600080C84BC0CA}</serialnumber>
            <version>1.0.0</version>
         </formid>
         <saveformat>application/vnd.xfdl</saveformat>
         <label>Time Sheet</label>
         <bgcolor>
            <ae>ivory</ae>
         </bgcolor>
         <fontinfo>
            <ae>Helvetica</ae>
            <ae>10</ae>
            <ae>plain</ae>
         </fontinfo>
      </global>
   </globalpage>
```

These global settings specify that:

- The form is written in XFDL version 6.0.
- All saves activated from the form should save the form as an XFDL form, unless specified otherwise in an item that initiates a save.
- The title Time Sheet should appear in the title bar of all pages, unless specified otherwise in a page global.
- All pages, toolbars and boxes should have an ivory background, unless they contain an option specifying otherwise.
- All pages and items should use a plain, Helvetica, 10-point font, unless they contain an option specifying otherwise. (Note: Labels that are parts of other items, like fields, are excluded from the fontinfo option. They are set using the labelfontinfo option, which is not available at this level.)

Usage Notes

1. Define form globals within the *global* item of the *globalpage*.
2. The *global* item and the *globalpage* must always have a sid of *global*.
3. The *globalpage* follows the XFDL tag.
4. You can give the form a title that appears in the title bar by setting a global *label* option.
5. When referencing form globals from within the form, the following syntax applies:

   `global.global.`*`option[n|name]`*

   For example:

   `global.global.triggeritem`

6.2 Page Globals

Page globals specify settings (like *next* and *saveformat*) and characteristics (like *bgcolor*) for the page within which they appear. Page globals appear within a *global* item at the top of each page definition, and apply to the whole page. They can be overridden by option settings within items.

Available Options

activated, bgcolor, border, focused, focuseditem, fontcolor, fontinfo, itemfirst, itemlast, keypress, label, mouseover, next, previous, pagefirst, pageid, pagelast, pagenext, pageprevious, printbgcolor, printfontcolor, printsettings, saveformat, transmitformat

Note: *For descriptions,see section "8. Details on Options and Array Elements".*

Example

The following example shows page globals on two pages within a single form:

```
<page sid="PAGE_1">
   <global sid="global">
      <pageid>
         <serialnumber>{D5D7080C84BC0CA}</serialnumber>
      </pageid>
      <printsettings>
         <borderwidth>1</borderwidth>
      </printsettings>
      <bgcolor>
         <ae>gray84</ae>
      </bgcolor>
      <label>Administration Form 1</label>
      <next>FIELD_date</next>
   </global>
   . . .

<page sid="PAGE_2">
   <global sid="global">
      <pageid>
         <serialnumber>{G5D7183C84DB0AA}</serialnumber>
      </pageid>
      <printsettings>
         <borderwidth>1</borderwidth>
      </printsettings>
      <bgcolor>
         <ae>gray84</ae>
      </bgcolor>
      <label>Administration Form 2</label>
```

```
    <next>FIELD_adminname</next>
</global>
. . .
```

“PAGE_1” has a medium-gray background, and directs the focus to the item called “FIELD_date” as soon as it opens. It assumes the rest of its settings from the form global. (If no form global exists, the page will assume the XFDL defaults.)

On “PAGE_2”, the focus is directed to the item called “FIELD_adminname” as soon as the page opens. “PAGE_2” assumes the rest of its settings from its page global and XFDL defaults.

Usage Notes

1. Page globals are defined in the *global* item at the top of a page, after the page declaration.
2. The *global* item must have a sid of *global*.
3. Page globals apply only to the page they are on.
4. Page globals are optional.
5. To specify a title to appear in the page’s title bar, use the label option as a page global.
6. When referencing page global options within the form, the following syntax applies:

   `pageid.global.option[n|name]`

   For example:

   `PAGE_1.global.bgcolor[0]`

7. Details on Items

Items are the basic elements of a page. The syntax of an item definition is as follows:

```
<itemType sid="itemTag">
   option definition1
   ...
   option definitionn
</itemType>
```

Notes:

- The itemType states the type of item to create. It must be one of the item types defined in this specification, or must be a custom item that follows the rules for custom items outlined in this specification.
- The *sid* attribute is mandatory.
- The value of each item sid must be unique in the page.

The *sid* attribute uniquely identifies an item. (See section “2.6 Scope Identifiers (sid)”.) Every item tag in a page must be unique. The *ItemType* element identifies the type of item to create. (For example, <*field*...> defines the item as a field.) This section contains information about XFDL-defined item types and the options available for each.

Note: *Defining an option more than once in an item’s definition is not permitted. See section “8. Details on Options and Array Elements” for descriptions of each option type.*

7.1 action

Specifies form-initiated actions that execute automatically. The actions can be any of the following types: link, replace, submit, done, display, print, refresh, pagedone, save, select, enclose, extract, remove, signature, or cancel. See section “8.84 type” for a description of each of these actions.

Action items can be defined to occur only once or repeat at specified time intervals, and after the page opens but before the page appears. See the section on the delay option for information on timing options. *Action* items can trigger either background actions or actions involving user interaction. A form can contain only hidden items such as *action* items and operate in the background. Such forms are called daemon forms.

Available Options

activated, active, data, datagroup, delay, itemnext, itemprevious, printsettings, transmitdatagroups, transmitformat, transmitgroups, transmititemrefs, transmititems, transmitnamespaces, transmitoptionrefs, transmitoption, transmitpagerefs, type, url

Example

The following action will send a status message to the server. The transaction happens automatically every 10 minutes (600 seconds).

```
<action sid="sendStatus_action">
   <delay>
      <ae>repeat</ae>
      <ae>600</ae>
   </delay>
   <type>submit</type>
   <url>
      <ae>mailto:manager@company.com</ae>
   </url>
</action>
```

Usage Notes

1. Repeating automatic actions is one method of creating a sparse-stated connection. It allows the form to indicate periodically to a server application that it is still running. Use the delay option to specify repetition.
2. Actions, by the form definition rules, reside on a page; therefore, actions occur only when the page is open, and repeating actions stop when the page closes. Actions defined to occur before the page displays, occur each time the page opens.

7.2 box

Specifies a rectangular box on the form. Other items may be positioned on top of boxes (using itemlocation). The purpose of *box* items is simply to add visual variety to the form.

Available Options

bgcolor, border, fontinfo, itemlocation, itemnext, itemprevious, printbgcolor, size, visible

Usage Notes

1. To make the box more visible, assign a background color that differs from the page background color (the default).
2. When setting the *size* option of a box, the height and width of the box will be based on the average character size for the font in use (set with the *fontinfo* option). The default font, if none is specified in the page global settings, is Helvetica 8 plain.

Example

The following example shows a typical box description. The box is 25 characters wide and 4 characters high. The background color is blue.

```
<box sid="blue_box">
   <bgcolor>
      <ae>blue</ae>
   </bgcolor>
   <size>
      <ae>25</ae>
      <ae>4</ae>
   </size>
</box>
```

7.3 button

Specifies a click button that performs an action when clicked with the mouse or activated with the space bar when it receives the input focus. Buttons can request data from a web server, submit or cancel the form, sign the form, save it to disk, or enclose external files.

Available Options

acclabel, activated, active, bgcolor, border, coordinates, data, datagroup, focused, fontcolor, fontinfo, format, help, image, imagemode, itemlocation, itemnext, itemprevious, justify, keypress, linespacing, mouseover, next, printbgcolor, printfontcolor, signature, signatureimage, signdatagroups, signdetails, signer, signformat, signgroups, signitemrefs, signitems, signnamespaces, signoptionrefs, signoptions, signpagerefs, size, transmitformat, transmititemrefs, transmititems, transmitoptionrefs, transmitoptions, transmitpagerefs, type, url, value, visible

Examples

Submit button

Buttons that trigger form processing requests must have a *type* option setting of **submit** or **done**. The definition for such a button might look like this:

```
<button sid="submit_button">
   <value>Process Form</value>
   <fontinfo>
      <ae>Helvetica</ae>
      <ae>18</ae>
      <ae>bold</ae>
      <ae>italic</ae>
   </fontinfo>
   <type>done</type>
   <url>
      <ae>http://www.server.com/cgi-bin/formProcessor</ae>
   </url>
</button>
```

Enclosure button

This button encloses an external file in the form. The action to enclose a file is enclose. The *datagroup* option identifies the list of datagroups, or folders, in which the user can store the enclosed file. An enclose button might take the following form:

```
<button sid="enclose_button">
   <value>Enclose File</value>
   <fontinfo>
      <ae>Helvetica</ae>
```

```
        <ae>18</ae>
        <ae>bold</ae>
        <ae>italic</ae>
    </fontinfo>
    <type>enclose</type>
    <datagroup="array">
        <ae>Images_Asia</ae>
        <ae>Images_Eur</ae>
        <ae>Images_SAmer</ae>
    </datagroup>
</button>
```

This button will allow users to enclose files into one of three datagroups (folders): Images_Asia, Images_Eur, Images_SAmer.

Usage Notes

1. The text in the button is defined by the *value* option. If no *value* option exists, the button will contain no text.
2. When setting the *size* option of a button, the height and width of the button is based on the average character size for the font in use (set with the *fontinfo* option).
3. If a button's image *option* points to a data item that does not exist or contains no data, then the button will display its *value* option instead.
4. If a button's *image* option points to a data item that dynamically changes its *mimedata* (but not its item tag), then the button will update the image it displays. For information on how to update an image by enclosing a new one, see the *data* option description.
5. The *format* option is available in buttons in order to force users to sign forms before submitting them.
6. There are two steps to making a signature button mandatory:
   - Assign the following elements to the *format* option: **string** and **mandatory**.
   - Set the button's *value* equal to the button's *signer* option setting.
   - Setting the *format* to **mandatory** specifies that the button must have a *value* setting that is not empty before the user submits the form. Equating the *value* to the setting of the *signer* option ensures that the only way a button's *value* is set is if somebody uses it to sign the form. (The *signer* option stores the identity of the person who signed the form using the button.)

Usage Notes: Signature Buttons

1. A signature button is the means by which the user can sign a form. To make a button a signature button, set its *type* to **signature**.

2. A signature button can be set up to sign the whole form or just part of it by setting up filters on the signature, using the *signdatagroups*, *signgroups*, *signitemrefs*, *signitems*, *signoptionrefs*, and *signoptions* options.

Note: *It is recommended that the* triggeritem *and* coordinates *options should be filtered out. These options change when a submission is triggered or when a user clicks an image button, respectively. Filtering out parts of the form that a subsequent user will change, including subsequent signatures and signature buttons and custom options that might change , should also be taken into consideration.*

---

3. Signature buttons allow users to do the following:
   - Sign the form or portion of the form the button specifies.
   - Delete their signatures (a signature can be deleted only by the user whose signature it is, and if the signature is currently valid and not signed by some other signature).
   - View the signature and view the XFDL text of what the signature applies to.
4. All option references, calculations, and other formulas in any signed portion of a form are frozen once they have been signed. Their setting will be valued at the setting they contained at the moment when the signature was created. If the user deletes the signature, however, then the formulas will become unfrozen, and will change dynamically as normal.
5. It is **strongly recommended** that you consider the usage notes in the *signitemrefs*, *signoptionrefs* and *signdatagroups* filters, and avoid using the other advanced filters unless a thorough security review has been performed.

   The usual options for other buttons (i.e. *size*, *image*, *value*) can also be used with signature buttons.

7.4 cell

Populates *combobox*, *list* and *popup* items. A cell can belong to multiple comboboxes, lists and popups. See the *combobox*, *list* and *popup* item sections for information on associating cells with these items.

Cells fall into two categories according to their behavior:

- **Action cells** — These cells perform the same set of actions normally associated with buttons. This includes such things as canceling, saving, and submitting the form.
- **Select cells** — These cells provide users with a mutually exclusive set of values from which to choose. When chosen, these cells appear selected. In a list this means the cell is highlighted in some way. In a popup, the cell's label becomes the popup's label.

Available Options

activated, active, data, datagroup, group, label, itemnext, itemprevious, printsettings, saveformat, transmitdatagroups, transmitformat, transmitgroups, transmititemrefs, transmititems, transmitoptionrefs, transmitoptions, transmitpagerefs, type, url, value

Example

The following example shows a list with three cells. To learn how to get the value of the user's selection, see Usage Notes below.

```
<popup sid="CountryPopup">
   <label>Country</label>
   <group>country</group>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
</popup>
<cell sid="albCell">
   <value>Albania</value>
   <group>country</group>
   <type>select</type>
</cell>
<cell sid="algCell">
   <value>Algeria</value>
   <group>country</group>
   <type>select</type>
</cell>
<cell sid="banCell">
   <value>Bangladesh</value>
   <group>country</group>
   <type>select</type>
</cell>
```

Usage Notes

1. Use the *type* option to establish a cell's behavior. Select cells have a type of *select* (the default type).
2. Cells can have both *value* and *label* options. These options affect the form differently depending on whether the cell is linked to a *combobox*, a *popup*, or a *list* item. In general, the *label* of the cell will be displayed as a choice, while the *value* of the cell will be displayed if that cell is selected. For more information, refer to the appropriate item type.
3. Cells take their color and font information from the *combobox*, *list* and *popup* items with which they are associated. In this way, a cell's appearance can vary according to the list the user is viewing.
4. To get the value of a cell that a user has selected from a list, it is necessary to dereference it in the following manner:

   ```
   page_tag.list_tag.value->value
   ```

   For example:

   ```
   page1.countryPopup.value->value
   ```

   When a user selects a cell from a list, the item tag of the cell is stored as the value of the list. Hence the dereference syntax.
5. Starting with version 4.5, items of type *cell* are the only items that can be filtered using the *signgroups* and *transmitgroups* options.

7.5 check

Provides a simple check box to record a selected or not selected answer from a user. A selected check box appears filled while a deselected box appears empty.

The exact appearance of the check box is platform-dependent; but the shape is rectangular. The check box appears as a normal check box for the users of each platform.

Available Options

acclabel, active, bgcolor, editstate, focused, fontcolor, fontinfo, help, itemlocation, itemnext, itemprevious, keypress, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, previous, printbgcolor, printfontcolor, printlabelbgcolor, printlabelfontcolor, saveformat, size, value, visible

Example

This *value* option setting in this check box is **on**, so the check box will appear selected when it displays. The item's label is Activate Health Plan, and the label will display in a Times 14 Bold font colored blue.

```
<check sid="healthPlan_check">
   <value>on</value>
   <label>Active Health Plan</label>
   <labelfontinfo>
      <ae>Times</ae>
      <ae>14</ae>
      <ae>bold</ae>
   </labelfontinfo>
   <labelfontcolor>
      <ae>blue</ae>
   </labelfontcolor>
</check>
```

Usage Notes

1. The *value* option setting indicates the user's answer. If the user selects or checks the check box, the *value* option contains **on**, otherwise it contains **off**. The default value is **off**.
2. Check boxes do not belong to groups like radio buttons — each check box may be turned on or off independently of the others.
3. The *label* option defines the label for the check box. The label appears above the check box and aligned with the box's left edge. There is no default label.
4. When setting the *size* option of a check box, the height and width of the bounding box will be based on the average character size for the font in use (set with the *fontinfo* option).
5. The *fontcolor* option determines the color of the check box fill pattern (default is **red**).

7.6 combobox

Comboboxes act like a hybrid of a field and a popup. Unopened, a combobox with a label occupies the same space as two labels, and a combobox without a label occupies the same space as a single label. After a user chooses a cell, the combobox closes (that is, returns to its unopened state).

If none of the cells are appropriate, the user can type other information into the combobox. When information is typed in, it is stored in the *value* option of the combobox. When a cell is selected, the *value* option stores the *value* of that cell, so unlike other list items (popup and list), dereferencing is not necessary.

A combobox's label appears above the *combobox* item.

Available Options

acclabel, activated, active, bgcolor, borderwidth, editstate, focused, fontcolor, fontinfo, format, group, help, itemlocation, itemnext, itemprevious, keypress, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, previous, printbgcolor, printlabelbgcolor, printfontcolor, printlabelfontcolor, size, value, visible

Example

This is an example of a combobox containing a set of selections allowing users to choose a color.

```
<combobox sid="CATEGORY_POPUP">
   <group>combo_Group</group>
   <label>Choose a Color:</label>
</combobox>
```

The default label is "Choose a Color:". This will display above the combobox. Until the user types in something or makes a selection, the field area of the combobox will be blank.

These are the cells that make up the combobox. They are select cells and they belong to the same group as the combobox: combo_Group.

```
<cell sid="RED_CELL">
   <group>combo_Group</group>
   <type>select</type>
   <value>Red</value>
</cell>
<cell sid="WHITE_CELL">
   <group>combo_Group</group>
   <type>select</type>
   <value>White</value>
</cell>
<cell sid="BLUE_CELL">
   <group>combo_Group</group>
   <type>select</type>
```

```
    <value>Blue</value>
</cell>
```

Usage Notes

1. Place cells in a combobox by creating a group for the combobox and assigning cells to the group. Create a group using the *group* option in the combobox definition. Assign cells to the group using the *group* option in the cell definition.
2. When first viewed, a combobox will display its *value*. If no value is set, the combobox will be empty.
3. The *label* option sets the text displayed above the item, as with a field.
4. When a combobox is opened, its list displays:
   - The *label* of each cell.
   - The *value* of each cell that does not have a *label*.
5. When a cell is selected from the combobox's list, the following occurs:
   - If the cell is of *type* **select**, the combobox's *value* is set to the *value* of the selected cell and the combobox displays that *value*.
   - If the cell is of any other *type*, the appropriate action for the type is taken and the combobox's *value* is set to empty.
6. When a value is typed into a combobox (rather than selected from the list), the combobox's *value* is set to the typed value.
7. The *label* option for each cell is used to create a long form for each choice. For example, a cell might have a *label* of "United States of America" and a *value* of "USA". The long form is displayed in the combobox's list, but once that cell is selected, the short form is displayed by the popup.
8. *Combobox*, *popup*, and *list* items with the same group reference display the same group of cells.
9. Unlike popups and lists, comboboxes do not need to be dereferenced in order to obtain the *value*.
10. When setting the *size* option of a combobox, the height and width of the popup will be based on the average character size for the font in use (set with the *fontinfo* option).
11. When setting the *editstate* option, the combobox will behave in the following manner:
    - A **readwrite** setting will cause it to function normally.
    - A **readonly** setting will cause the combobox to refuse all input, although it will function normally otherwise and formulas will still be able to change the *value*.
    - A **writeonly** setting will cause the combobox to use "password" characters in its field contents, but the list of choices will still be displayed in plain text.
12. When a *format* is applied to a combobox, the formatting will be applied to the *value* of each cell linked to the combobox. Those cells that fail the check will be flagged or filtered. Those cells that

pass the check will have their *value* replaced with a formatted value. See the *format* option for more information.

If any two *combobox*, *list*, or *popup* items use the same set of cells, they must apply the same formatting.

7.7 data

Stores an information object such as an image, a sound, or an enclosed file in an XFDL form. Whenever any of these objects are are added to a form, the data that describes the object is stored in a *data* item. A *data* item can only store the data from a single object. Data in *data* items must be encoded in **base64** format.

Data items are created automatically when files are enclosed in a form. Enclose files using items with a *type* option setting of **enclose**.

Available Options

datagroup, filename, itemnext, itemprevious, mimedata, mimetype

Example

This is an example of a *data* item.

```
<data sid="Supporting_Documents_1">
   <mimetype>text/plain</mimetype>
   <filename>HelloWorld.txt</filename>
   <mimedata encoding="base64">
       SGVsbG8sIHdvcmxkIQ==
   </mimedata>
   <datagroup>
      <ae>Supporting_Documents</ae>
   </datagroup>
</data>
```

Usage Notes

1. Stores the data in the *mimedata* option, and the data's MIME type in the *mimetype* option.
2. If a *data* item contains a *datagroup*option, it can be associated with one or more other data items. *Data* items with a *datagroup*option are not replaced if a button or cell of type **enclose** point to new *data* items. The new *data* items simply become members of the same *datagroup*. Additionally, buttons and cells with the same *datagroup*option can access the contents of the *data* item.
3. If a button or cell of type **enclose** contains a *data* option that points to a *data* item (as opposed to using the *datagroup* option), then special rules apply to the *data* item's behavior. If a user encloses a new *data* item using that button, the new information overwrites the old. For example, if the *data* item originally contained a jpeg image of a dog, and then a user enclosed a png image of a house, then the *data* item's *mimedata*, *mimetype*, and *filename* options update themselves to contain the information about the house image.
4. Starting with XFDL version 4.5, *data* items are the only items that can be filtered using the *signdatagroups* and *transmitdatagroups* options.

7.8 field

The *field* item creates a text area where users can display and enter one or more lines of data. The field's characteristics determine the number of lines, the width of each line, and whether the field is scrollable.

Field data can be protected from modification, made to display in the system password format (typically, hidden from view), and forced to conform to data type and formatting specifications.

Available Options

acclabel, active, bgcolor, border, editstate, focused, fontcolor, fontinfo, format, help, itemlocation, itemnext, itemprevious, justify, keypress, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, previous, printbgcolor, printlabelbgcolor, printfontcolor, printlabelfontcolor, rtf, scrollvert, scrollhoriz, size, texttype, value, visible

Example

This is an example of a single line *field* item that allows 20 characters of input. An initial value of 23000 has been defined for the field. When the form appears, the field will contain this value.

```
<field sid="income_field">
   <label>Annual income</label>
   <value>23000</value>
   <size>
      <ae>20</ae>
      <ae>1</ae>
   </size>
   <fontinfo>
      <ae>Courier</ae>
      <ae>12</ae>
      <ae>plain</ae>
   </fontinfo>
   <labelfontinfo>
      <ae>Helvetica</ae>
      <ae>12</ae>
      <ae>plain</ae>
   </labelfontinfo>
   <labelfontcolor>
      <ae>blue</ae>
   </labelfontcolor>
</field>
```

Usage Notes

1. When setting the *size* option of a field, the width of the field will be based on the average character size for the font in use (set with the *fontinfo* option) and the height will exclude the external font leading.
2. The *editstate* option determines whether the field is read only, write only (for passwords, for example), or available for both reading and writing.
3. The *format* option specifies the data type of the field's data. It also contains flags that allow the application of specified edit checks and formatting to the data.
4. The *label* option defines the field's label. The label is placed above the field and aligned with the field's left edge.
5. The *scrollvert* and *scrollhoriz* options govern a field's scrolling characteristics. They must be set to always to permit scrolling. With scrolling enabled, scroll bars display along the bottom (horizontal scrolling) and right (vertical scrolling) edges of the field.

7.9 help

Defines a help message that can be used to support various external items in the form. Separate *help* items can be created for every item supported, or one *help* item can be used to support several items.

Available Options

active, itemnext, itemprevious, value

Example

This is an example of a *button* item that links to a *help* item using the *help* option:

```
<button sid="fullPicture_button">
   <value>View Full-Sized Picture</value>
   <help>button help</help>
   <fontinfo>
      <ae>Times</ae>
      <ae>14</ae>
      <ae>plain</ae>
   </fontinfo>
   <type>link</type>
   <url>
      <ae>http://www.server.com/application/fullPic.frm</ae>
   </url>
</button>
```

The following example shows the *help* item referred to in the *button* item definition. The contents of the *value* option are presented as the help message when the user asks for help with the button.

```
<help sid="button_help">
   <value>
      Pressing this button will bring a full-sized image in a form
      down to your viewer.
   </value>
</help>
```

Usage Notes

1. The *help* item's *value* option contains the help message text.
2. The link between the *help* item and the supported item is created by the *help* option in the supported item's definition. The *help* option contains the *help* item's item reference.

7.10 label

Defines a static text message or an image to display on the form. If both an image and a text message are defined for the label, the image takes precedence in viewers able to display images.

Available Options

active, bgcolor, , border, fontcolor, fontinfo, format, help, image, imagemode, itemlocation, itemnext, itemprevious, justify, linespacing, printbgcolor, printfontcolor, size, value, visible

Example

This is an example of a multiple-line text label:

```
<label sid="RHYME LABEL">
   <value>
      Little miss Muffet Sat on her tuffet,
      Eating her curds and whey.
      When along came a spider, who sat down beside her,
      and frightened miss Muffet away!
   </value>
   <fontinfo>
      <ae>Times</ae>
      <ae>16</ae>
      <ae>italic</ae>
   </fontinfo>
   <justify>right</justify>
</label>
```

Usage Notes

1. To define the text for a label, use the *value* option. To define an image for a label, use the *image* option.
2. To create a multiple line text message, add line breaks to the message text. Use the escape sequence '\n' to indicate a line break.
3. When setting the *size* option of a label, the height and width of the label will be based on the average character size for the font in use (set with the *fontinfo* option).
4. If a label's image *option* points to a data item that does not exist or contains no data, then the label will display its *value* option instead.
5. If a label's *image* option points to a *data* item that dynamically changes its *mimedata* (but not its item tag), then the label will update the image it displays. For information on how to update an image by enclosing a new one, see the *data* option description.
6. Set the image display behavior with the *imagemode* option.

7. The label's background color defaults to being transparent — and thus the label will allow whatever item it is over to show through. For example, it is possible to place a label over another label holding an image. The image will show through the top label.
8. Label contents (if text) can be formatted using *format* flags (see section "8.18 format").

7.11 line

Draws a simple vertical or horizontal line on the form. Lines are useful for visually separating parts of a page.

Available Options

fontcolor, fontinfo, itemlocation, itemnext, itemprevious, printfontcolor, size, thickness, visible

Example

This is an example of a horizontal line with a thickness of five pixels:

```
<line sid="BLUE_LINE">
   <size>
      <ae>40</ae>
      <ae>0</ae>
   </size>
   <thickness>5</thickness>
</line>
```

Usage Notes

1. Specify the dimensions of a line using the *size* and *thickness* options. The *size* option determines whether the line is vertical or horizontal. If the horizontal dimension is set to zero, then the line is vertical. If the vertical dimension is set to zero, then the line is horizontal. Size is calculated in characters.
2. The *thickness* option determines how thick the line will be. Thickness is calculated in pixels.
3. The *fontinfo* option information is used when calculating the line's size. The *size* option's unit of measurement is characters; therefore, choice of font can affect the size. See the *size* option for more information.
4. The *fontcolor* option defines the color of the line.

7.12 list

Creates a list from which users can make selections (as in a list of names) and trigger actions (such as enclosing files and submitting the form). A list can contain both selections and actions.

The entries in the list are *cell* items. Selections are cells with a *type* option setting of **select**. Actions are cells with any other *type* option setting.

Available Options

acclabel, active, bgcolor, border, editstate, focused, fontcolor, fontinfo, format, group, help, itemlocation, itemnext, itemprevious, keypress, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, previous, printbgcolor, printlabelbgcolor, printfontcolor, printlabelfontcolor, size, value, visible

Example

This is an example of a list containing three actions: submit form, save form, and cancel form.

```
<list sid="MAINMENU_LIST">
   <group>list_Group</group>
   <label>Options Menu</label>
   <labelfontcolor>
      <ae>blue</ae>
   </labelfontcolor>
   <size>
      <ae>3</ae>
      <ae>20</ae>
   </size>
</list>
```

These are the cells that make up the list. They are action cells and they belong to the same group as the list: **list_Group**.

```
<cell sid="SUBMIT_CELL">
   <group>list_Group</group>
   <type>submit</type>
   <url>
      <ae>http://www.server.com/cgi-bin/processForm</ae>
   </url>
   <value>Submit Form</value>
</cell>
<cell sid="SAVE_CELL">
   <group>list_Group</group>
   <type>save</type>
   <value>Save Form</value>
</cell>
```

```
<cell sid="CANCEL_CELL">
   <group>list_Group</group>
   <type>cancel</type>
   <value>Cancel this Form</value>
</cell>
```

Usage Notes

1. Place cells in a list by creating a group for the list and assigning cells to the group. Create a group using the *group* option in the list definition. Assign cells to the group using the *group* option in the cell definition.
2. When first viewed, a list displays it's *label* above the item (as with a field) and as many choices as its size allows.
3. The choices in the list display:
   - The *label* of each cell.
   - The *value* of each cell that does not have a *label*.
4. When a cell is selected from the list, the following occurs:
   - If the cell is of *type* **select**, the list's *value* is set to the name of the selected cell and that cell is highlighted.
   - If the cell is of any other *type*, the appropriate action for the type is taken, the list's *value* is set to empty, and the selected cell is highlighted.
5. The *label* option for each cell is used to create a long form and and abbreviated form for each choice. For example, a cell might have a *label* of "United States of America" and a *value* of "USA". The long form is displayed in the list, but the short form available as the cell's *value*.
6. To get the value of a cell that a user has selected from a list, it is necessary to dereference it in the following manner:

   ```
   page_tag.list_tag.value->value
   ```

   For example:

   ```
   <compute>page1.countryPopup.value->value</compute>
   ```

7. *List*, *combobox* and *popup* items with the same group reference display the same group of cells.
8. The *value* option will contain one of the following:
   - The item reference of the most recently chosen cell, if the cell was of type **select**.
   - Nothing, if the cell most recently chosen was of any type other than **select**.
9. When setting the *size* option of a list, the height and width of the list will be based on the average character size for the font in use (set with the *fontinfo* option).

10. A vertical scroll bar will appear beside the list if the number of cells is greater than the height (defined with the *size* option) of the list.
11. When a *format* is applied to a list, the formatting will be applied to the value of each cell linked to the list. Those cells that fail the check will be flagged or filtered. Those cells that pass the check will have their *value* replaced with a formatted *value*. See the *format* option for more information.
12. If any two *combobox*, *list*, or *popup* items use the same set of cells, they must apply the same formatting.

7.13 popup

Creates a popup menu from which users can make selections (as in a list of names) and trigger actions (such as enclosing files and submitting the form). A popup can contain both selections and actions.

The entries in the popup are *cell* items. Selections are cells with a *type* option setting of **select**. Actions are cells with any other *type* option setting.

Popups act like a hybrid of a label, a button, and a list. Unopened, a popup occupies only the space required for its label. Open, the popup displays a list of selections and actions. After a user chooses a selection or an action, the popup closes (that is, returns to its unopened state). A popup's label displays inside the *popup* item.

Available Options

acclabel, activated, active, bgcolor, border, editstate, focused, fontcolor, fontinfo, format, group, help, itemlocation, itemnext, itemprevious, justify, keypress, label, mouseover, next, previous, printbgcolor, printfontcolor, size, value, visible

Example

This is an example of a *popup* list containing a set of selections allowing users to choose a category.

Here is the *popup* definition. The default label is "Choose a Category:". This will display until a user makes a selection. Afterwards, the cell's value will display as the label.

```
<popup sid="CATEGORY_POPUP">
   <group>popup_Group</group>
   <label>Choose a Category:</label>
</popup>
```

These are the cells that make up the popup. They are select cells and they belong to the same group as the popup: **popup_Group**.

```
<cell sid="HISTORY_CELL">
   <group>popup_Group</group>
   <type>select</type>
   <value>World History</value>
</cell>
<cell sid="SCIENCE_CELL">
   <group>popup_Group</group>
   <type>select</type>
   <value>Physical Sciences</value>
</cell>
<cell sid="MUSIC_CELL">
   <group>popup_Group</group>
   <type>select</type>
   <value>Music</value>
</cell>
```

Usage Notes

1. Place cells in a popup by creating a group for the popup and assigning cells to the group. Create a group using the *group* option in the popup definition. Assign cells to the group using the *group* option in the cell definition.
2. When first viewed, a popup will display its *label* if no *value* is set. If there is no *value* or *label*, the popup will be empty.
3. When a popup is opened, its list displays:
   - The *label* of each cell.
   - The *value* of each cell that does not have a *label*.
4. When a cell is selected from the popup's list, the following occurs:
   - If the cell is of *type* **select**, the popup's *value* is set to the name of the selected cell and the popup displays the cell's *value*.
   - If the cell is of any other *type*, the appropriate action for that type is taken, the popup's *value* is set to empty, and the popup displays its *label* option
5. The *label* option for each cell is used to create a long form for each choice. For example, a cell might have a *label* of "United States of America" and a *value* of "USA". The long form is displayed in the popup's list, but once that choice is selected, the short form is displayed by the popup.
6. To get the *value* of a cell that a user has selected from a list, it is necessary to dereference it in the following manner:

   ```
   page_tag.list_tag.value->value
   ```

   For example:

   ```
   <compute>page1.countryPopup.value->value</compute>
   ```

   When a user selects a cell from a list, the item tag of the cell is stored as the *value* of the list. Hence the dereference syntax.
7. *Popup*, *combobox* and *list* items with the same group reference display the same group of cells.
8. When setting the *size* option of a popup, the height and width of the popup will be based on the average character size for the font in use (set with the *fontinfo* option).
9. When a *format* is applied to a popup, the formatting will be applied to the value of each cell linked to the popup. Those cells that fail the check will be flagged or filtered. Those cells that pass the check will have their *value* replaced with a formatted *value*. See the *format* option for more information.
10. If any two comboboxes, lists, or popups use the same set of cells, they must apply the same formatting.

7.14 radio

Intended for use with one or more other *radio* items. A group of radio buttons presents users with a set of mutually exclusive choices. Each radio button represents one choice the user can make.

There is always one selected radio button in the group. As well, since radio buttons present a mutually exclusive set of choices, only one radio button in a group can be selected. When a user chooses a radio button, that radio button becomes selected.

A selected radio button appears filled in some way. All other radio buttons in the group appear empty.

Available Options

acclabel, active, bgcolor, editstate, focused, fontcolor, fontinfo, group, help, itemlocation, itemenxt, itemprevious, keypress, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, previous, printbgcolor, printlabelbgcolor, printfontcolor, printlabelfontcolor, size, value, visible

Example

This example shows a group of three radio buttons. The first radio button is the initial choice: the *value* option setting is on. The buttons all belong to the group **search_Group**.

```
<radio sid="NAME_RADIO">
   <value>on</value>
   <group>search_Group</group>
   <label>Search by Name</label>
</radio>

<radio sid="NUMBER RADIO">
   <group>search_Group</group>
   <label>Search by Number</label>
</radio>

<radio sid="OCCUPATION RADIO">
   <group>search_Group</group>
   <label>Search by Occupation</label>
</radio>
```

As shown here, only the chosen radio button needs to have a *value* option setting. The remaining radio buttons will receive the (default) value setting of **off**.

Usage Notes

1. Group radio buttons by assigning them to the same group. Do this by including the *group* option in each radio button's definition, and using the same group reference in each case.
2. The *value* option contains the status indicator. It can be either **on** or **off**. The value **on** indicates a status of chosen. The value **off** indicates a status of not chosen. The default status is not chosen.

3. When the form opens, if no radio button has the status chosen, then the last radio button defined for the group becomes chosen. If multiple radio buttons are chosen, then only the last ‘chosen’ radio button retains that status.
4. The *label* option defines a label to appear above the radio button and aligned with its left edge.
5. When setting the *size* option of a radio button, the height and width of the bounding box will be based on the average character size for the font in use (set with the *fontinfo* option).
6. The *fontcolor* option determines the color of the radio button fill pattern (default is red).

7.15 signature

Contains a signature and the data necessary to verify the authenticity of a signed form. It is created by a form viewer or other program when a user signs a form (usually using a signature button). The signature item contains an encrypted hash value that makes it impossible to modify the form without changing the hash value that the modified form would generate. To verify, one can generate the hash value and then see if it matches the one in the signature.

Available Options

excludedmetadata, fullname, itemnext, itemprevious, layoutinfo, mimedata, signature, signdatagroups, signdetails, signer, signformat, signgroups, signitemrefs, signitems, signnamespaces, signoptionrefs, signoptions, signpagerefs

Example

This example shows a signature item below the signature button that created it.

```
<button sid="empSigButton">
   <type>signature</type>
   <value compute="signer"></value>
   <signer>Jane D Smith, jsmith@insurance.com</signer>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <signformat>application/vnd.xfdl;
      csp="Microsoft Base CryptographicProvider v1.0";
      csptype=rsa_full;hashalg=sha1
   </signformat>
   <signoptions>
      <ae>omit</ae>
      <ae>triggeritem</ae>
      <ae>coordinates</ae>
   </signoptions>
   <signitemrefs>
      <ae>omit</ae>
      <ae>PAGE1.mgrSigButton</ae>
      <ae>PAGE1.admSigButton</ae>
      <ae>PAGE1.empSignature</ae>
      <ae>PAGE1.mgrSignature</ae>
      <ae>PAGE1.admSignature</ae>
   </signitemrefs>
   <signature>empSignature</signature>
</button>
...
<signature sid="empSignature">
```

```
    <signformat>application/vnd.xfdl;
        csp="Microsoft Base CryptographicProvider v1.0";
        csptype=rsa_full;hashalg=sha1
    </signformat>
    <signer>Jane D Smith, jsmith@insurance.com</signer>
    <fullname>
        "VeriSign, Inc.", VeriSign Trust Network,
        "www.verisign.com/repository/RPA Incorp. by
        Ref.,LIAB.LTD(c)98", Persona Not Validated,
        Digital ID Class 1 - Microsoft, Jane D
        Smith, jsmith@insurance.com
    </fullname>
    <signature>PAGE1.empSignature</signature>
    <signitemrefs>
        <ae>omit</ae>
        <ae>PAGE1.mgrSigButton</ae>
        <ae>PAGE1.admSigButton</ae>
        <ae>PAGE1.empSignature</ae>
        <ae>PAGE1.mgrSignature</ae>
        <ae>PAGE1.admSignature</ae>
    </signitemrefs>

<!-- The items listed above MUST have itemlocation options with
    absolute and extent as the last settings in order for the filter
    below to be sufficient in terms of security -->

    <signoptionrefs>
        <ae>keep</ae>
        <ae>PAGE1.mgrSigButton.itemlocation</ae>
        <ae>PAGE1.admSigButton.itemlocation</ae>
        <ae>PAGE1.empSignature.itemlocation</ae>
        <ae>PAGE1.mgrSignature.itemlocation</ae>
        <ae>PAGE1.admSignature.itemlocation</ae>
    </signoptionrefs>

    <signoptions>
        <ae>omit</ae>
        <ae>triggeritem</ae>
        <ae>coordinates</ae>
    </signoptions>
    <mimedata encoding="base64">
        MIIFMgYJKoZIhvcNAQcCoIIFIzCCBR8CAQExDzANBgkgAQUFADALB\ngk
        qhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl6hMrc5DySSP+X5j\nANf
        BGSOI\n9w0BAQQwDwYDVQQHEwhJbn<Rlcm5ldDEXMBUGA1UEChM\nOVmV
        yaVNpZ24sIEluYy4xNDAKn1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmR
        dWFsIFN1YnNjcmliyZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCAR
        ExETA
    </mimedata>
</signature>
```

Usage Notes

1. When a user signs a form using a signature button, the viewer creates the *signature* item as specified in the button's *signature* option. The viewer also associates the signature with the signature button, using the signature's *signature* option.
2. When a user signs a form, the *signer*, *signformat*, *signgroups*, *signitemrefs*, *signitems*, *signnamespaces*, *signoptionrefs*, and *signoptions* options are copied from the button description to the signature description.
3. While *signformat* is not mandatory for *button* items, it is mandatory for signature items.
4. A copy of the XFDL description of the form or portion of the form that is signed is included in the signature's *mimedata* option. This data is encrypted using the hash algorithm specified in the button's *signformat* option.
5. When a program checks a signed form, it compares the data in the *mimedata* option with that of the portion of the form that is apparently signed. If the descriptions match, then the signature remains valid. If the signatures do not match, the signature breaks, and the user is prompted.
6. An attempt to create a signature will fail if:
   - The item named by the signature button's *signature* option already exists.
   - The signature button is already signed by any signature in the form.
   - The signer's private key is unavailable for signing.
7. Filters can be used to indicate which items and options to keep and to omit. The explicit and implicit settings of an existing filter take precedence over an implication that might be drawn from a non-existing filter. Set up these filters in the signature button description.
8. To use certain types of digital signatures (CryptoAPI or Netscape, for example), it is necessary for the user to obtain a digital signature certificate. Other types of digital signatures require the user to have a pen/pad device installed on the user's computer.

7.16 spacer

Creates space between items on a form. It can be any size specified. It is invisible.

Available Options

fontinfo, itemlocation, itemnext, itemprevious, label, linespacing, size

Example

This example shows a *spacer* item that uses the *size* option to define the amount of space it will occupy.

```
<spacer sid="THREE_SPACER">
   <size>
      <ae>1</ae>
      <ae>3</ae>
   </size>
</spacer>
```

This example shows the *spacer* item that uses a label to define the amount of space it will occupy. This sizing technique is useful when creating a spacer that is exactly the same size as a real label on the form.

```
<spacer sid=”WELCOME_SPACER”>
   <label>Welcome to Information Line</label>
</spacer>
```

Usage Notes

1. A spacer can be sized either by giving it length and width dimensions (using *size*), by expanding the default size using the *itemlocation* option or by giving it a *label*. If a *label* is used, the spacer equals the size of the text typed into the label. The label does not appear; it is simply used to determine the spacer’s size.
2. When setting the *size* option of a spacer, the height and width of the spacer will be based on the average character size for the font in use (set with the *fontinfo* option).

7.17 toolbar

Allows the definition of a toolbar for a page. A toolbar is a separate and fixed area at the top of the page. It functions much like a toolbar in a word processing application. Items placed in the toolbar are always visible at the top of the form, no matter what portion of the page they are viewing.

The toolbar is visible no matter what portion of the page body is visible. However, if the toolbar is larger than half the form window, it is necessary to scroll to see everything it contains.

Available Options

bgcolor, itemnext, itemprevious, mouseover

Example

This example shows a toolbar that contains a label.

Here is the toolbar definition:

```
<toolbar sid="TOOL_BAR">
   <bgcolor>
      <ae>cornsilk</ae>
   </bgcolor>
</toolbar>
```

Here is the label that will appear in the toolbar.

```
<label sid="COMPANY_NAME">
   <value>My Company</value>
   <itemlocation>
      <ae>
         <ae>within</ae>
         <ae>TOOL_BAR</ae>
      </ae>
   </itemlocation>
</label>
```

Usage Notes

1. The background color of the toolbar becomes the default background color for items in the toolbar.
2. Add items to the toolbar using the **within** modifier of the *itemlocation* option. Code the *itemlocation* option in each included item's definition.
3. Each page can contain only one toolbar.

7.18 <custom item>

Allow form designers to add application specific information to the form definition. This is useful when submitting forms to applications requiring non-XFDL information. An example of non-XFDL information might be an SQL query statement.

Available Options

All XFDL options and any custom options can be used with custom items.

Example

This is an example of a custom item definition. It includes both an XFDL and a custom option.

```
<custom:event xfdl:sid="STATUS_EVENT"
       xmlns:custom="http://www.PureEdge.com/XFDL/Custom">
   <xfdl:active>off</xfdl:active>
   <custom:ID>UF45567/home/users/preferences01</custom:ID>
</custom:event>
```

Usage Notes

1. Custom items must be in a non-XFDL namespace.
2. Custom items can also change the default namespace from the XFDL namespace URI in order to eliminate excess prefixes within the desired namespace.
3. Custom items may omit the XFDL scope identifier, but a *sid* attribute in the XFDL namespace must be provided in order to reference the custom item's content with the XFDL compute system. Since default namespaces are not applied to attributes, the *sid* attribute must still be qualified by a namespace prefix associated with the XFDL namespace URI.
4. Both XFDL options and custom options within the custom item can have computed values by using the XFDL compute attribute, which must be qualified with a namespace prefix associated with the XFDL namespace URI.

8. Details on Options and Array Elements

Syntax

For simple character data content:

```
<optionTag>character data content</optionTag>
```

For computed options:

```
<optionTag compute="expression">character data content</optionTag>
```

For array options:

```
<optionTag>
   <!-- suboption elements -->
</optionTag>
```

An option defines a characteristic given to a form, a page, or an item. For example, the *bgcolor* option set at the form or page global level defines the background color of the pages of the form itself whereas a *bgcolor* option set at the item level defines the background color for the containing item. Some form and page global options define defaults for item-level options. For example, if an item has no fontinfo option, then the fontinfo in the page globals are used, and if the page globals contain no *fontinfo* option, then the *fontinfo* in the form globals (and the form global *fontinfo* has implied defaults if it is not specified).

The definition of an option consists of content between start and end tags. The element tag defines the type of option. This type must be one of the option types defined in this specification, or a user-defined option that follows the rules in the “custom option” description in this specification.

Option Content

The content of an option can take one of three formats: simple character data, a compute, or an array of subordinate XML elements. Computes are identified by a compute attribute, while arrays are identified by the presence of subordinate elements.

Simple Character Data

The default is simple character data, in which case the option must contain text with no child elements. For example:

```
<value>This is the value</value>
```

Compute

If the character content must be computed, then the computational expression appears in the start tag of the option in an attribute named *compute*. If the XFDL computation system has been applied to the

form, then the the option also contains simple character data for the current computed value of the expression. For example:

```
<value compute="price1Field.value + price2Field.value * ’0.07’"
   >205.68</value>
```

It is typical to have a form run its computes on a client machine, then have server modules simply read the current values, ignoring the content of the compute attributes. In essence, an application can ignore the compute attributes unless it must change element values that are referenced by computed options. See section “3. The XFDL Compute System” for details on how the compute expression is represented.

Array

The third case for an option’s content is an array of subordinate elements. The option must contain one or more array elements. For example:

```
<bgcolor>
   <ae>255</ae>
   <ae>248</ae>
   <ae>220</ae>
</bgcolor>
```

Each array element may also contain an array. This recursive definition permits arbitrary depth for XFDL arrays.

Array Element Names

A number of the XFDL-defined options use array elements. XFDL does not often assign names to the array elements, so the default tag name of *ae* is used. Some XFDL options have named array elements, so the XML tag name is not *ae* (for a list of named array elements, see Rule [24] in “2.3 Options and Array Elements”).

Order of Precedence of Options

An option set at a lower level in the form hierarchy overrides a similar option set at a higher level. It overrides it for only the level it is in and any that come below it in the hierarchy. For example, the *fontinfo* option in the following example would override a global *fontinfo* setting for the page it is in, and also for any items in that page.

```
<page sid="Page1">
   <global="global">
      <fontinfo>
         <ae>Helvetica</ae>
         <ae>12</ae>
         <ae>plain</ae>
      </fontinfo>
   </global>
```

Defining Form Global and Page Global Options

Form global options are optional and must be defined in a <global> element in a <globalpage> element after the XFDL start tag and before the first <page> in a form. Page global options are optional and must be defined in a <global> element after the <page> start tag and before the first item in a page. To determine whether an option is a valid form global or page global option, see section “6. Global Settings”.

Data Type Designators Used in Option Descriptions

XFDL defines a set of data types that describe type of content allowed in an option. Each option description in this specification uses one or more of the following data type designators:

Data Type	Description
char	A single ASCII character.
string	A series of ASCII characters.

color	A color name or an RGB triplet representing the color. The syntax of an RGB triplet is:
	`<bgcolor>`
	`  <ae>`*`red value (0 to 255)`*`</ae>`
	`  <ae>`*`green`* `value (0 to 255)</ae>`
	`  <ae>`*`blue`* `value (0 to 255)</ae>`
	`</bgcolor>`
	
	For example, the triplet for green is:
	`<ae>0</ae>`
	`<ae>255</ae>`
	`<ae>0</ae>`

coordinate	Whole number in the range 0 to 1,000 representing one coordinate of a position.
integer	Positive or negative whole number in the range -32,768 to 32,767.
unsigned byte	Whole number in the range 0 to 255.
unsigned	Whole number in the range 0 to 65,535.
numeric boolean	A value of 0 or 1, in which 0 is false and 1 is true.

8.1 acclabel

Defines a message that is available to active screen readers. When the focus shifts to the item containing the acclabel, the message is read aloud by the screen reader. The message should contain additional information about the item to assist users with vision impairments.

Syntax

```
<acclabel>message</acclabel>
```

message	*string*	a message that is read aloud to users

Available In

button, check, combobox, field, list, popup, radio

Example

The following example shows a field that contains an accessibility message.

```
<field sid="firstName">
   <acclabel>Type your first name.</acclabel>
</field>
```

When the focus moves to the field, the acclabel message will be passed to the screen reader, which will read “Type your first name” aloud.

Usage Notes

1. Default: **none**
2. Screen readers normally provide information in addition to the *acclabel* option. This information is defined by the screen reader in use, and cannot be controlled through XFDL.
3. If the item contains a *label* option, the screen reader will read the *label* option as well as the *acclabel* option.
4. If the item has a *value* option, the screen reader will read the item’s value. If the item has a blank value, the screen reader will say that the item is empty.
5. If the item has associated cells, the screen reader reads the *value* option of the cells.
6. If the item contains a *help* option, the screen reader will read the *value* of the associated help item. The screen reader will also read this help message if the item contains a *format* option and the item’s content is invalid when the user tries to tab to a new item

7. If the item contains a *format* option with a **message** flag, the screen reader will read the contents of the **message** flag if the item's content is invalid when the user tries to tab to a new item.
8. If the item is signed, the screen reader reads the signed message as well as the *acclabel* option. The signed message is defined by the Viewer and is not modifiable through XFDL.
9. If you are using a *label* item to provide information for another item, the *acclabel* option of the second item should include the contents of the *label* item. Since *label* items never receive the focus, screen readers will never read their contents. The *acclabel* option can provide the same information to users, with further explanation if necessary.
10. The following table provides the actions which invoke the screen reader and the order in which it reads the messages provided by the form and the Viewer:

Action	Viewer Help On	Item is Signed	Messages Read by Screen Reader
Item gains focus	No	No	label option + acclabel + value
Item gains focus	Yes	No	label option + acclabel + value + help
Item gains focus	n/a	Yes	label option + acclabel + value + help + signed
Tab out of item failed	n/a	n/a	format + help
Spacebar or arrow keys activate list of choices	n/a	No	cell value

8.2 activated

Specifies whether an item, page, or form is currently activated by the user or not. This option is usually set by an external program such as a parser, but under certain circumstances can be set by computes in the form. The *activated* option for an item can be set by a compute if the item is an action, button or cell utilized on the current page. The *activated* option must be set to either **on** or **off** and the item must be capable of being activated. Cells that are not grouped with a *popup*, *combobox* or *list* on the current page cannot be activated. Buttons that are not visible cannot be activated.

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<activated>status</activated>
```

status	*on*	item, page, or form is currently activated by user
	off	item, page, or form is not currently activated by user
	maybe	button only: item might be activated, as user has pressed it, but has not yet released it

Available In

action, button, cell, combobox, popup, page global, form global

Example

The following example shows a button that changes color when it is activated:

```
<button sid="saveButton">
   <type>save</type>
   <value>Save</value>
   <activated>off</activated>
   <bgcolor>
      <ae compute="toggle(activated, 'off', 'on') ? 'white' :
         'gray'"></ae>
   </bgcolor>
</button>
```

The button will appear white when the user activates it, and gray otherwise.

The following example shows how the activated option can be set for an item based on user input. This use of the activated option works with the keypress option to establish a default button on the form.

```
<button sid="DefaultButton">
   <type>cancel</type>
   <activated>off</activated>
   <opt_1 compute="toggle(global.keypress, '', 'ESC') == '1'  &#xA;
      ? set('activated','on') : ''"></opt_1>
</button>
```

When the user presses the ESC key (which is not processed by any item on the form), the activated option for the button will be set to **on** and the button will fire.

Usage Notes

1. Default: **off**
2. *Activated* is set to **on** when an item is activated, and remains on until any transaction initiated by the item is properly under way. For example, in a print button, activated will be turned on when the user initiates the print action, and will remain on until network results indicate the print action is taking place.
3. The *activated* option is not included in form descriptions that are saved or transmitted.
4. Specific details on activated behavior for each item:
   - **action** — actions set *activated* to **on** when they fire, and **off** when the transaction they initiate is under way.
   - **button** — buttons set *activated* to **maybe** when the user holds the mouse pointer or space bar down on the button. They set it to **on** if the user releases the pointer or space bar while over the button, and they set activated to **off** when the transaction the button initiates is under way.
   - **cell** — cells behave in the same manner as buttons. In the split second during which a user selects a select type of cell, it sets *activated* to **on**. It turns *activated* **off** as soon as the action of being selected is finished. Cells that initiate network transactions set *activated* to **on** from the beginning of the request to the time when the request produces results. Note that there is no maybe status for a cell.
   - **combobox and popup** — comboboxes and popup lists set *activated* to **on** when their lists are popped open, and **off** when the lists are not open. Note that the “field” portion of a combobox does not register an *activated* setting.
   - **page** — a page sets *activated* to **on** while it is open, and **off** when it is not. A page can be open and activated even when the form is minimized (not actively on screen).
   - **form** — a form sets *activated* to **on** while it is open, and **off** when it is not. A form can be open and activated even when it is minimized (not actively on screen) .

8.3 active

Specifies whether an item is active or inactive. Inactive items do not respond to user input and, if possible, appear dimmed. For example, an inactive check box will be dimmed and the user will not be able to select or deselect the box.

Syntax

```
<active>status</active>
```

status	*on*	item is active
	off	item is inactive

Available In

action, button, cell, check, field, help, label, list, popup, radio

Example

This sample specifies the item is active.

```
<active>on</active>
```

Usage Notes

1. Default: **on**
2. Setting active to off would be similar to setting an editstate of readonly.

8.4 bgcolor

Defines the background color of a page or an item.

Syntax

```
<bgcolor>
   RGB triplet
</bgcolor>
```

or

```
<bgcolor>
   <ae>color name<ae>
</bgcolor>
```

Note:

- Either format is acceptable.

color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See “Data Type Designators Used in Option Descriptions” on page 85 for the syntax of an RGB triplet.

Available In

box, button, check, combobox, field, label, list, popup, radio, toolbar, page global, form global

Examples

These samples both set the background color to forest green.

```
<bgcolor>
   <ae>forest green</ae>
</bgcolor>

<bgcolor>
   <ae>34</ae>
   <ae>139</ae>
   <ae>34</ae>
</bgcolor>
```

Usage Notes

1. Default: varies depending on the object
2. The transparent color has no RGB equivalent.
   - Form: **white**
   - Page: the form bgcolor setting or default (white)
   - Item (depends on the item type):
     - button items: **gray** (or grey)
     - check, combobox, field, list, popup, and radio items: **white**
     - label items: **transparent** (if no color is specified, the label background color will be the same as the page background color).
     - All other items: the background color of the page.

8.5 borderwidth

Defines whether an item is displayed with a border. Borders are drawn as a three dimensional effect.

Syntax

```
<borderwidth>width</borderwidth>
```

width	*numeric boolean*	**zero** for no border or **one** for a border

Available In

box, button, combobox, field, label, list, popup, page global, form global

Example

This sample sets the item to display a border:

```
<borderwidth>1</borderwidth>
```

Usage Notes

1. **Default**:
   - For *label* items, the default is always **zero**, regardless of global page or form settings.
   - For all other items, the default is the global page setting, or global form setting if there is no page setting, or **one** if there are no global settings.

8.6 coordinates

Records the position of the mouse pointer on an image. The image must exist in a button item. The recording occurs when a user selects (i.e. clicks) the button using the mouse pointer.

The position is an intersection on an unseen grid overlaying the image. The points along each axis of the grid range from zero (0) through 1000 with position 0,0 occurring in the button's top left corner. The coordinates map the intersection closest to the mouse pointer's position.

Syntax

```
<coordinates>
   <ae>X_coordinate</ae>
   <ae>Y_coordinate</ae>
</coordinates>
```

X_coordinate	*coordinate*	the coordinate on the X axis
Y_coordinate	*coordinate*	the coordinate on the Y axis

Available In

button

Example

When a user clicks on a button containing an image, a *coordinates* option statement is inserted into the *button* item definition. The statement would look something like this. This particular setting indicates a position at the intersection of points 180 on the x-axis and 255 on the y-axis.

```
<coordinates>
   <ae>180</ae>
   <ae>255</ae>
</coordinates>
```

Usage Notes

1. Default: **none**

8.7 data

Associates an *action*, *button*, or *cell* item with a single *data* item. The *data* option is valid only in items with a type setting of **enclose**, **display**, **extract**, or **remove**.

Syntax

```
<data>data item</data>
```

data item	*string*	the item tag of the data item to associate with the action, button, or cell

Available In

action, button, cell

Example

The button below is an enclosure button associated with a single data item.

```
<button sid="encloseImageButton">
   <value>Update Image</value>
   <type>enclose</type>
   <data>displayImage</data>
</button>
```

If a user enclosed another file, then the *data* item referred to in the button's *data* option would be replaced with the new *data* item. (The *data* item would use the same item tag — the one that's referred to in the *data* option.)

Usage Notes

1. Default: **none**
2. A *data* option specifies only zero or one data items.
3. If an item with a type setting of **enclose** and a *data* option is used to enclose a second *data* item, then the second *data* item will replace the first.
4. If an enclosure mechanism is used to replace an image stored in a *data* item with a new image (see above), then buttons and labels whose *image* option is set to the identifier of the image *data* item will be updated to display the new image.
5. A *data* item referred to in a *data* option might also have a *datagroup* option and thus belong to the datagroups of other actions, buttons, or cells.

8.8 datagroup

Provides a way of associating related *data* items to each other and to certain other items. There are two ways of using this option. In the first case, it enables you to create a group of *data* items, called a datagroup. In the second case, this option enables you to reference such a datagroup from *button*, *action*, or *cell* items.

This option is most often used to group file enclosures. For example, you can use this feature to create folders with which users can organize their enclosures. Each enclosed file can belong to several datagroups, and each datagroup can contain several enclosed files.

Syntax

```
<datagroup>
   <ae>datagroup reference1</ae>
   ...
   <ae>datagroup referencen</ae>
</datagroup>
```

where *datagroup* reference is one of:

- datagroup name for datagroups on the current page
- page *tag.datagroup name* for datagroups on other pages

Note:

- Include a *datagroup reference* entry for each datagroup this item accesses.

datagroup reference	*string*	identifies a data group

Available In

action, button, cell, data

Example

If this sample were part of a data item definition, it would mean the data item belonged to the datagroups **Business_Letters**, **Personal_Letters**, and **Form_Letters**.

If this sample were part of an *action*, *button*, or *cell* item, it would mean the user could store the enclosure in one of the three datagroups.

```
<datagroup>
   <ae>Business_Letters</ae>
```

```
    <ae>Personal_Letters</ae>
    <ae>Form_Letters</ae>
</datagroup>
```

In the following example, the Enclose button references the datagroups **Business_Letters** and **Personal_Letters**. As a result, when users click this button, they can choose to place the file they are enclosing into one of these folders. Because the datagroup **Form_Letters** is not specified in the *datagroup reference*, it is not available to “BUTTON1”.

```
<button sid="BUTTON1">
    <value>Click to Enclose File</value>
    <type>enclose</type>
    <datagroup>
        <ae>Business_Letters</ae>
        <ae>Personal_Letters</ae>
    </datagroup>
</button>
```

Usage Notes

1. Default: **none**
2. The grouping of *data* items into datagroups cannot span multiple pages. That is, all the *data* items assigned to a given datagroup must belong to the same page. On the other hand, buttons, actions, and cells from one page can reference datagroups from another page, provided that you specify the page tag in the datagroup reference.
3. Used with items handling enclosures, datagroup lists the datagroups the item can access. Used with a *data* item, datagroup lists the datagroups to which the enclosure belongs. Enclosures are stored in *data* items.
4. Items that handle enclosed files perform **enclose**, **extract**, **remove**, and **display** actions. These actions types are set using the *type* option.
5. When a user selects an item that handles enclosed files, the list of datagroups appears. The user chooses the datagroup (or folder) with which to work. If the action is enclosing, the enclosed file is added to that datagroup. Otherwise, a list of files in the datagroup appears. The user chooses a file from the list.
6. The action of enclosing a file creates the *data* item, and stores the user’s choice of *datagroup* (or folder) in the data item’s *datagroup* option.

8.9 delay

Delays the execution of an automatic action or specifies an automatic action repeat factor. Repeated actions stop when the page containing the action definition closes. Define automatic actions using an *action* item.

Syntax

```
<delay>
   <ae>repeat factor</ae>
   <ae>interval</ae>
</delay>
```

repeat factor	*repeat*	queue the action to repeat at the <interval> specified
	once	perform the action once after the <interval> specified
interval	*integer*	the frequency of repeated actions or the delay before performing single occurrence actions. The unit of measurement is seconds.
	-1	perform the action before the page displays. Only valid with a repeat factor of once.

Available In

action

Example

This sample sets the action to occur once, 15 minutes (900 seconds) after the page opens.

```
<delay>
   <ae>once</ae>
   <ae>900</ae>
</delay>
```

Usage Notes

1. Defaults:
   - repeat factor: **once**
   - interval: **zero seconds**

2. This means the action will occur when the page appears.
3. Repeating automatic actions is one method of creating a sparse-stated connection. It allows the form to indicate periodically to a server application that it is still running.
4. All actions with the same interval occur in the order they are defined in the page.
5. The page does not display while actions with an interval of **-1** are running.

8.10 dirtyflag

Alerts the form viewing program to the changed status of the form. If the user has altered any portion of the form since the last save, the dirtyflag will be set to on, and any subsequent attempts by the user to close the form will cause the viewing program to display a save prompt. The dirtyflag can be set to **off** if desired, and the user will be able to close the form without the viewing program displaying a save prompt. The dirtyflag can also be used for internal checks on the changed status.

Syntax

```
<dirtyflag>status<dirtyflag>
```

status	*on*	indicates that the form has been changed since the last save or network submission
	off	indicates that the form has not been changed since the last save or network submission

Available In

form global

Example

This example shows how the dirtyflag can be set to off.

```
<button sid="ApplicationSpecificSave">
   <value>Save</value>
   <type>cancel</type>
   <opt_1 compute="toggle(activated, 'off', 'on') == '1' ? &#xA;
      MySaveFunction() + set('global.global.dirtyflag', &#xA;
      'off') : ""></opt_1>
</button>
```

Usage Notes

1. Defaults: **none**
2. Use the *set* function to clear the dirtyflag.

8.11 editstate

Defines one of three possible edit states for modifiable items.

Syntax

```
<editstate>edit state</editstate>
```

editstate	*readonly*	users cannot change the item's setting
	writeonly	users can change, but not see, the item's setting
	readwrite	users can see and change the item's setting

Available In

check, combobox, field, list, popup, radio

Example

This sample sets the editstate to readonly.

```
<editstate>readonly</editstate>
```

Usage Notes

1. Default: **readwrite**.
2. The **writeonly** setting applies only to fields and comboboxes. It causes all characters the user types to appear the same as the system password character.
3. The **readonly** setting permits users to scroll an item even though they cannot update the item's contents.

8.12 excludedmetadata

This option allows additional data about a signature to be included, but never signed. This makes it possible to store the notarization of signatures without interfering with other, overlapping signatures.

For example, if Signer1 signs a form and then Signer2 affixes an overlapping signature, you could not modify the first signature without breaking the second. In this case, you would not be able to notarize the first signature, since affixing the notarization would change the mimedata of that signature and break the second signature.

The *excludedmetadata* provides a place to store the notarization for the first signature without breaking the second signature. You can add information to this option at any time, since the *excludedmetadata* option is never signed.

Syntax

```
<excludedmetadata>
   <servernotarizations>
      <ae>Notarization1</ae>
      ...
      <ae>Notarizationn</ae>
   </servernotarizations>
</excludedmetadata>
```

Notarization	*string*	a compressed base64 encoded PKCS-7 signature that signs the hash of the mimedata option and the details of the signature that is being notarized

Available In

signature

Example

The following example shows an *excludedmetadata* option with two notarizing signatures. Note that the base64 blocks would be much larger in practice.

```
<excludedmetadata>
   <servernotarizations encoding="base64-gzip">
      <ae>asdfkj439fgasdf81hgb</ae>
      <ae>opkbt1ed7f8y3476p294</ae>
   </servernotarizations>
</excludedmetadata>
```

Usage Notes

1. Default: **none**

8.13 filename

Identifies the name of an enclosed file. This name appears in the list of enclosed files.

Syntax

```
<filename>name of file</filename>
```

name of file	*string*	the name of the enclosed file

Available In

data

Example

This sample specifies the name of an enclosed file:

```
<filename>std_logo.xfd</filename>
```

Usage Notes

1. Default: **none**
2. To ensure cross-platform compatibility, limit filenames to the following set of characters: lowercase letters from a to z, uppercase letters from A to Z, the integers 0 through 9, and the underscore (_).
3. To ensure cross-platform compatibility, limit form names to a maximum of eight characters, followed by a *.xfd* extension.

8.14 focused

Specifies whether an item, page, or form currently has the input focus. This option is usually set by code outside XFDL, but can also be set by a compute, provided that the compute is setting the focus of an item to **on**, the item is on the same page, and the item receiving the focus is capable of doing so.

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<focused>status</focused>
```

status	*on*	item, page, or form has input focus
	off	item, page, or form does not have input focus

Available In

button, check, combobox, field, list, popup, radio, page global, form global

Example

The following example shows a button that changes its color to white if it has the input focus, and to blue if it does not.

```
<button sid="saveButton">
   <type>save</type>
   <value>Save</value>
   <focused>off</focused>
   <bgcolor>
      <ae compute="focused==’on’ ? ’white’ : ’blue’"></ae>
   </bgcolor>
</button>
```

The following example shows how the focus can be moved to a different item based on user input.

```
<check sid="CHECK1">
   <value>off<value>
   <label>Check here to skip next section</label>
   <opt_1 compute="toggle(value, ’off’, ’on’) == ’1’ ? &#xA;
         set(’FIELD14.focused’,’on’): ’’"></opt_1>
</check>
```

Usage Notes

1. Default: **off**
2. The *focused* option is set to **on** when an item, page, or form receives the input focus, and is set to **off** when the focus is moved to another item, page or form.
3. An object's *focused* option does not change when the form application displaying it becomes active or inactive on a desktop. For example, a page that is open on screen will have a *focused* option set to **on**, even if the page is minimized or is not the currently active application on the desktop.
4. In objects that are hierarchical, it is possible for more than one object to have the focus at one time. For example, a form, a page, and a field can all be focused at the same time.
5. When a form viewing application is closing a form, it should set all *focused* options to **off** and then resolve all formulas before shutting down.
6. The *focused* option is not included in form descriptions that are saved or transmitted.

8.15 focuseditem

Specifies which item in the page currently has the focus.

Syntax

```
<focuseditem>sid</focuseditem>
```

sid	*string*	the sid of the item that currently has the focus

Available In

page global

Example

The following example show the page global when the nameField item has the focus.

```
<page sid="PAGE1">
   <global sid="global">
      <focuseditem>nameField</focuseditem>
   <global>
   <field sid="nameField">
      <value>John Doe</value>
   </field>
</page>
```

Usage Notes

1. When the page changes, the focuseditem for the previous page is set to empty. For example, if the user changed from “PAGE1” to “PAGE2”, the *focuseditem* option for “PAGE1” would be set to an empty string.
2. The *focuseditem* option is maintained by the form viewing application, and is not included in form descriptions that are saved or transmitted.

8.16 fontcolor

Defines the font color for the text or filler portion of an item. In *radio* and *check* items, fontcolor defines the color of the bullet and check, respectively. In *line* items, fontcolor defines the color of the line. In other items, it defines the text color.

Syntax

```
<fontcolor>
   <ae>color name</ae>
</fontcolor>
```

or

```
<fontcolor>
   RGB triplet
</fontcolor>
```

Note:

- Either format is acceptable.

color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See “Data Type Designators Used in Option Descriptions” on page 85 for the syntax of an RGB triplet.

Available In

button, check, combobox, field, label, line, list, popup, radio, page global, form global

Examples

These samples both set the font color to chocolate.

```
<fontcolor>
   <ae>chocolate</ae>
</fontcolor>

<fontcolor>
   <ae>210</ae>
   <ae>105</ae>
```

```
    <ae>30</ae>
</fontcolor>
```

Usage Notes

1. Defaults behave as follows:
   - If there is no fontcolor setting at the item level, the item will derive its *fontcolor* setting from the page or form global level. If no fontcolor setting is specified at the page or form level, the default is **black**.
   - For *check* and *radio* items, the default is not inherited from the page or global level, and is always **red**.

8.17 fontinfo

Defines the character set, font name, point size, and font characteristics for the text portion of an item. Note that the font selected for an item influences the item's size.

Syntax

```
<fontinfo>
   <ae>character set</ae>
   <ae>font name</ae>
   <ae>point size</ae>
   <ae>weight</ae>
   <ae>effects</ae>
   <ae>form</ae>
   <ae>character set</ae>
</fontinfo>
```

Note:

- weight, effects, form, and character set are optional.

font name	*string*	the name of the font
point size	*unsigned byte*	the size of the font
weight	*plain*	use plain face
	bold	use bold face
effects	*underline*	underline the text
form	*italics*	use the italic form
character set	*string*	the name of the character set

Available In

box, button, check, combobox, field, label, line, list, popup, radio, spacer, page global, form global

Example

This sample sets the font information to Times 14, bold italic in the ANSI character set:

```
<fontinfo>
   <ae>
```

```
    <ae>Times</ae>
    <ae>14</ae>
    <ae>bold</ae>
    <ae>italic</ae>
</fontinfo>
```

Usage Notes

1. Defaults:
   - font name: **Helvetica**
   - point size: **8**
   - weight: **plain**
   - effects: **not underlined**
   - form: **not italics**
   - character set: **ANSI**
2. If any of the fontinfo settings are invalid, then the defaults will be used.
3. The *size* option calculates item size using the font's average character width. Therefore, choice of font affects item width.
4. XFDL supports the following fonts, font sizes, and character sets:
   - Fonts: Courier, Times, Symbol (symbol), Helvetica, and Palatino.
   - Sizes: 8, 9, 10, 11, 12, 14, 16, 18, 24, 36, 48.
   - Character Sets: ANSI and Symbol.
5. Other fonts and font sizes may be used. However, especially for cross-platform Internet applications, it is best to choose from the ones cited above since they are guaranteed to work.

8.18 format

Allows edit checks and formatting options for *field*, *label*, *list*, *popup*, and *combobox* items to be specified. It allows a **mandatory** status for signature *button* items to be specified (for details, see the button item description).

Syntax

```
<format>
   <ae>data type</ae>
   <ae>format flag</ae>
   <ae>check flag</ae>
</format>
```

Notes:

- Multiple flags are valid.
- Data type is mandatory and must appear first; the flags are optional and may appear in any order.

data type	*(see below)*	the type of data the field must contain
format flag	*(see below)*	the type of formatting applied to the user's input
check flag	*(see below)*	the type of edit check performed on the formatted input

Available In

button, combobox, field, label, list, popup

Data Types [Types]

XFDL supports the following data types:

Data Type	Description	Format Defaults To:
`<ae>string</ae>`	free form character data up to 32K long	any character
`<ae>integer</ae>`	a positive or negative whole number in the range of -2,147,483,648 to +2,147,483,647	any whole number

Data Type	Description	Format Defaults To:
`<ae>float</ae>`	a positive or negative floating point decimal number in the range of 1.7 * 10^{-308} to 1.7 * 10^{308}	any decimal number
`<ae>dollar</ae>`	a fixed point decimal number with a scale of 2 and a range equal to the range of a float	any number. Automatically adds .00 to end, if no decimal value specified.
`<ae>date</ae>`	a date including day-of-month, month, and year	this format: 3 Mar 1996
`<ae>day_of_week</ae>`	the name or number of a day of the week	this format: Thu
`<ae>month</ae>`	the name or number of a month	this format: Mar
`<ae>day_of_month</ae>`	the number of a day of the month	number format
`<ae>year</ae>`	a numeric year designation	this format: 1996 | 2000 B.C.
`<ae>time</ae>`	a time value containing hours and minutes from the 12 hour or the 24 hour clock	this format: 11:23 P.M.
`<ae>void</ae>`	disable entire format option (including data type, checks, and formats)	no effects on contents of a field

Format Flags

Any number of format flags in a format line can be specified. To see which format flags apply to each data type, see the cross reference table at the end of this section.

The available format flags are:

Format Flag	Description
`<ae>comma_delimit</ae>`	Delimit the thousands by commas.
`<ae>space_delimit</ae>`	Delimit the thousands by spaces.
`<ae>bracket_negative</ae>`	Indicate negative values by surrounding the value with parentheses, that is ( ).
`<ae>add_ds</ae>`	Add a dollar sign to the start of the value (dollar fields only).

Format Flag	Description
`<ae>upper_case</ae>`	Convert alphabetic characters to upper case.
`<ae>lower_case</ae>`	Convert alphabetic characters to lower case.
`<ae>title_case</ae>`	Convert first letter of each word to upper case and all other letters to lower case, for titles and proper names.
`<ae>short</ae>`	Uses a “short” format based on the data type: ■ **day_of_week** — numeric value in range 1 to 7 where 1 represents Sunday ■ **day_of_month** — numeric value in range 1 to 31 month - numeric value in range 1 to 12 ■ **year** — apostrophe followed by last two digits in year (‘98), ‘before Christ’ era designator is B.C. (‘98 B.C.) Note that using this format with this data type may result in non-compliance with Year 2000 guidelines ■ **date** — year as four digits, month as two digits, and day-of-month as two digits, organized in YMD order (1998-04-29) ■ **time** — 24 hour clock (as in 23:30)
`<ae>long</ae>`	Uses a “long” format based on the data type: ■ **day_of_week** — name in full (as in Monday) ■ **day_of_month** — two digits plus suffix (as in 01st) ■ **month** — name in full (as in January) ■ **year** — four-digit numeric format; “before Christ” era designator is B.C. (as in 1549 B.C.) ■ **date** — long day_of_month, long month, long year formats, in DMY order (as in 22nd March 2000) ■ **time** — 12-hour clock with the time-of-day suffix, A.M. or P.M. (as in 9:39 P.M.)

Format Flag	Description
`<ae>numeric</ae>`	Display dates and times using numeric values and, possibly, the minus sign: ■ **day_of_week** — 2 digits in range 01 to 07 where 01 represents Sunday ■ **day_of_month** — 2 digits in range 01 to 31 ■ **month** — 2 digits in range 01 to 12 ■ **year** — 4 digits; “before Christ” era designator is minus sign (as in -1995) ■ **date** — month and day-of-month formats as above; year format is 4 digits; “before Christ” era designator is minus sign; organized in YMD order; no punctuation (as in 19980429 or -19980429) ■ **time** — 24 hour clock (as in 23.30)

`<presentation>`
`yy/mm/dd`
`</presentation>`
	Available only when formatting dates, to create custom template for presentation of dates, using Y for year, M for month, and D for day.
	For example:
	`<presentation>`
	`"YY/MM/DD"`
	`</presentation>`
	This would yield 98/12/23. Note that using this format with this data type may result in non-compliance with Year 2000 guidelines.

`<ae>void</ae>`	No formatting is applied.

Check Flags

Any number of edit checks in a format line can be specified. The edit checks specified and any edit checks implied by the data type will be performed.

To see which edit checks apply to each data type, see the cross reference table at the end of this section.

Note: *XFDL specifies that fields be formatted before an edit check is performed. For example, if the field’s data type is* ***dollar*** *and the* ***add_ds*** *and* ***comma_delimit*** *format options are specified, then the input 23000 becomes $23,000.00 before edit checks are applied. This can affect length and template checks. In this example, the length before formatting was 5 but it became 10 before edit checking*

The available check flags are:

Check Flag	Description
`<ae>optional</ae>`	Input from the user is not mandatory.
`<ae>mandatory</ae>`	Input from the user is mandatory.
`<ae>case_insensitive</ae>`	Apply edit checks without regard to the case in which the user enters the data.

`<range>`
`  <ae>low</ae>`
`  <ae>high</ae>`
`</range>`
	The field's value must be in the range specified. The range may be alphabetic, numeric, days of the week, days of the month, months, or years. Ranges cannot vary from high to low. For example, 10 to 1, the year 2000 to 1900, and so on, are invalid.

`<length>`
`  <ae>min</ae>`
`  <ae>max</ae>`
`</length>`
	Restrict the length of the formatted input data to a minimum of "min" bytes and a maximum of "max" bytes.

`<template>`
`  <ae>a</ae>`
`  <ae>b</ae>`
`  <ae>...</ae>`
`</template>`
	This is a list of templates permitted for the field. There is no restriction on the number of templates. Field contents must match one of the templates in the list. Any of the following wild card characters can be used:
	■ ? — represents any one (1) character
	■ * — represents any number of characters
	■ # — represents any one (1) numeric character
	■ % — represents any number of numeric characters
	■ @ — represents any one (1) alphabetic character
	■ ! — represents any number of alphabetic characters (which can include none)

`<message>help</message>`	Sets the error message that is displayed if the user input fails the type checking. The default message is, "This entry is invalid. Please try again."
`<ae>fail_checks</ae>`	Forces failure of format statement
`<ae>ignore_checks</ae>`	Causes all type checking checks to be ignored. Note: only checks are ignored, not formatting or data type.
`<ae>strip_meridiem</ae>`	Removes AM or PM from a time.

Cross Reference of Data Types, Format Flags, and Check Flags

Data Type	Applicable Format Flags	Applicable Check Flags
string	lower_case, upper_case, title_case	case_insensitive, fail_checks, ignore_checks, length, mandatory, message, optional, range, template
integer	bracket_negative, comma_delimit, space_delimit	fail_checks, ignore_checks, length, mandatory, message, optional, range, template
float	bracket_negative, comma_delimit, space_delimit	fail_checks, ignore_checks, length, mandatory, message, optional, range, template
dollar	add_ds, bracket_negative, comma_delimit, space_delimit	fail_checks, ignore_checks, length, mandatory, message, optional, range, template
date year month day_of_month day_of_week time	long, short, numeric	case_insensitive, fail_checks, ignore_checks, length, mandatory, message, optional, range, template
void	no formatting or type checking is done	no checking or type checking is done

Examples

This example specifies a field containing integer data with a range of values from 10 to 1,000 inclusive, and formatted with commas separating the thousands:

```
<format>
   <ae>integer</ae>
   <ae>comma_delimit</ae>
   <range>
      <ae>10</ae>
      <ae>1000</ae>
   </range>
</format>
```

This example specifies a field that contains dollar data that is mandatory. An error message appears if the data is not entered correctly.

```
<format>
   <ae>dollar</ae>
```

```
    <ae>mandatory</ae>
    <message>Entry incorrect--try again.</message>
</format>
```

This example specifies a field in which date data will be formatted as day-of-month, month, and year (i.e., 15th June 1999):

```
<format>
    <ae>date</ae>
    <ae>long</ae>
</format>
```

This example contains two templates. User input must match one of them:

```
<format>
    <ae>string</ae>
    <template>
        <ae>###-###-####</ae>
        <ae>###-###-####-###</ae>
    </template>
</format>
```

This example contains a decision: if a check box called “allowIncompleteCheck” is checked, then filling out the item is optional; if the check box is checked, then the item is mandatory and the user must complete it.

```
<format>
    <ae>string</ae>
    <ae compute="page1.allowIncompleteCheck==’on’ ? &#xA;
        ’optional’ : ’mandatory’"></ae>
</format>
```

Usage Notes

1. Default Formatting:
   - Case remains unchanged.
   - **Numeric value** format contains no thousands delimiter. This permits easy conversion of ASCII to integer format.
   - **Dollar value** format uses two decimal places and no dollar sign.
   - Zero is always positive.
   - **Day-of-week** and **month** format is the abbreviated name with no punctuation. For example, the 2nd day of the week is always *Mon*; the first month is always *Jan*.
   - The **year** format is long.
   - The **day_of_month** is short.

- The **date** format uses the default **day-of-month**, **month**, and **year** formats organized in DMY order as in 25 Dec 1995. The ‘before Christ’ era designator is B.C.
- The **time** format defaults to long. For example, 10:35 A.M. and 11:50 P.M.

2. Default Edit Checks:
   - All checking is case sensitive.
   - The default edit checks combine the requirements of the data type with any formatting requirements (default or specific).
   - If neither **optional** nor **mandatory** are specified, the rules that are specified will determine whether the user must enter information.
3. If a format flag conflicts with the data type, the format flag will be ignored.
4. All edit checks specified will be applied to the input data. This might result in a field the user cannot change. For example, the combination of data type **integer** and check flag *<template>a* </template>* creates such a situation. Data cannot be both an integer and begin with a letter.
5. While XFDL fully supports compliance with Year 2000 date formatting guidelines, it is possible to format dates in such a way that they are not Year 2000 compliant. Care should be taken to ensure that date formatting does not result in Year 2000 problems.
6. When applying a format to a combobox, list, or popup, the formatting will be applied to the value of each cell linked to the item. Those cells that do not pass the check will be flagged or filtered. If a cell passes the checks, its value will be replaced with a formatted value before the item is displayed. The *label* option for these cells will remain unaffected.
7. When applying a format to a *combobox*, *list*, or *popup* item, a cell with an empty value will fail all format checks but will still be selectable, even if input is mandatory. This allows users to erase their previous choice (which will also reset all formulas based on that choice). However, users will still need to select a valid cell before they can submit the form.
8. If any two comboboxes, lists, or popups use the same set of cells, they must apply the same formatting.
9. The **void** format type disables a format line completely through the use of a compute. **Void** formats never fail regardless of the checks in the format statement.
10. The **void** format flag facilitates the modification of a format statement by a formula. It is ignored by the formatting system.
11. For details on using the *format* option in buttons, see the Usage Notes in the button item description.

8.19 formid

Defines a unique identifier for the form.

Syntax

```
<formid>
   <title>form title</title>
   <serialnumber>serial number</serialnumber>
   <version>version number</version>
</formid>
```

Note:

- serialnumber and version are mandatory; title is optional

form title	*string*	provides a title for the form
serialnumber	*string*	provides a unique identifying string for the form. This is generated by an external program such as a form design program.
version number	*number in the format AA.Bb.cc*	shows the major (AA), minor (Bb) and maintenance (cc) numbers for the form. This is generated by an external program such as a form design program, but can also be set manually.

Available In

form global

Example

This sample shows how the formid option appears in the global characteristics of a form.

```
<formid>
   <title>Admin_Form</title>
   <serialnumber>{94EC8BA0-7D33-11D2-B5E3-0060}</serialnumber>
   <version>4.8.2</version>
</formid>
```

Usage Notes

1. Defaults:
   - title: **none**
   - serialnumber: **none**
   - version: **1.0.0**
2. This option is intended for use with form design programs and licensing models.

8.20 fullname

Used in a signature item to record the fully qualified name of the signer. This name is retrieved from the digital certificate used to the sign the form.

Syntax

```
<fullname>name</fullname>
```

name	*string*	the fully qualified name of the signer, as supplied by the digital certificate used to sign the form

Available In

signature

Example

This sample shows a fullname option as part of a signature item:

```
<signature sid="empSignature">
   <signer>Jane D Smith, jsmith@insurance.com</signer>
   <fullname>
      "VeriSign, Inc.", VeriSign Trust Network,
      "www.verisign.com/repository/RPA Incorp. by
      Ref.,LIAB.LTD(c)98", Persona Not Validated,
      Digital ID Class 1 - Microsoft, Jane D Smith,
      jsmith@insurance.com
   </fullname>
</signature>
```

Usage Notes

1. This option is added to the form during the signing process, and is created by the software that enables signing (such as a form viewer).

8.21 group

Provides a way of associating related items. There are two ways of using this option. In the first case, it enables you to to create groups of cells or radio buttons. In the second case, the *group* option enables you to populate lists, popups, and comboboxes by referencing a group of cells. Items with the same group reference are considered members of the same group.

Syntax

```
<group>group reference</group>
```

where group reference is one of:

- group name for groups on the current page
- page tag.group name for groups on other pages

group reference	*string*	identifies the group

Available In

cell, combobox, list, popup, radio

Example

In the following sample, the group option creates a group called “LIST1_GROUP” containing the cell items “CELL1” and “CELL2”.

```
<cell sid="CELL1">
   <group>LIST1_GROUP</group>
   <value>red</value>
   <type>select</type>
</cell>
<cell sid="CELL2">
   <group>LIST1_GROUP</group>
   <value>green</value>
   <type>select</type>
</cell>
```

In the following code, the *group* option is used to populate the list item with the cells from the the preceding sample.

```
<list sid="LIST1">
```

```
    <group>LIST1_GROUP</group>
    <label>Colors</label>
    <value></value>
</list>
```

Usage Notes

1. Default: **none**
2. The association of cells or radio buttons into groups cannot span multiple pages. That is, all the cells or radio buttons assigned to a given group must belong to the same page. On the other hand, you are allowed to populate a list, popup, or combobox with a group of cells defined on another page, as long as you specify the page tag in the group reference.
3. *List* and *popup* items are populated with cells that have the same group reference as the item. It is possible to have multiple *list* and *popup* items with the same group reference. In this way, the same group of cells can populate more than one list or popup.
4. All *radio* items having the same group reference will form a mutually exclusive group.

8.22 help

Points to the help message for the item. The *item reference* identifies the *help* item containing the help message. There can be many items pointing to the same help message.

Syntax

```
<help>item reference</help>
```

item reference	*string*	identifies the help item

Available In

button, check, combobox, field, label, list, popup, radio

Example

This sample points to the help item general_help defined on the page called page_1.

```
<help>page_1.general_help</help>
```

Usage Notes

1. Default: **none**

8.23 image

Associates an image with an item. The *item reference* identifies the *data* item containing the image. This image replaces any text label if the viewer is able to display images.

Syntax

```
<image>item reference</image>
```

item reference	*string*	identifies the data item

Available In

button, label

Example

This sample points to the data item company_logo defined on the page called page_lst.

```
<image>page_1st.company_logo</image>
```

Usage Notes

1. Default: **none**
2. Use this option to associate images with *button* and *label* items.
3. If an enclosure mechanism is used to replace an image stored in a *data* item with a new image, then buttons and labels whose *image* option is set to the identifier of the image data item will be updated to display the new image. For details, see the *data* option description.
4. Use the *imagemode* option to control the display behavior of the image.

8.24 imagemode

Defines how the image will be displayed in the item.

Syntax

```
<imagemode>image mode</imagemode>
```

image mode	*clip*	image is placed in the item from the top left corner. Parts of the image that extend past the item's bounding box are cut off.
	resize	image is placed in the item from the top left corner. The image is then expanded or contracted in both directions so that it fits the item exactly.
	scale	image is placed in the item from the top left corner. The image is expanded or contracted, keeping the original aspect ratio, to the point at which one of the sides fits snugly in the item, and the other side is within the item.

Available In

button, label

Example

This sample displays a company logo as a resized image.

```
<image>page_1st.company_logo</image>
<imagemode>resize</imagemode>
```

Usage Notes

1. Default: **resize**
2. An imagemode of **clip** draws the image in the upper left corner of the item. If the image is too big for the item's space, and the image is clipped at the item's edge. If the image is smaller than the item in either horizontally or vertically, the image is centered appropriately.
3. An imagemode of **resize** resizes the image to be the exact size of the item, whether that means an increase in size, decrease in size, or an increase in one dimension and decrease in the other.

4. An imagemode of **scale** also resizes the image to fit the item, except that it will preserve the aspect ratio of the original image. If either dimension of the image is larger than the item, then the image will be made small enough to fit in the given space as follows:
   - The larger dimension will fit snugly in the space.
   - The other dimension will be scaled by the same factor.

   Likewise, if both dimensions of the image are smaller than the space given by the button or label, then the image will be expanded to fit the given space as follows:
   - The larger dimension will fit snugly in the space.
   - The other dimension will be scaled by the same factor.
5. Use this option with the *image* option to control the image's appearance with button and label items.

8.25 itemfirst

Identifies the first item on the page, excluding the global item. An item is *first* when it appears first in the build order (in other words, it is first in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<itemfirst>item reference</itemfirst>
```

item reference	*string*	a reference to the first item on the page, excluding the global item.

Available In

page global

Example

This sample shows how the *itemfirst* option appears in the page globals of a form.

```
<page sid="PAGE1">
   <global sid="global">
      <itemfirst>LABEL1</itemfirst>
   </global>
</page>
```

Usage Notes

1. Defaults: **none**
2. The *itemfirst* option is not saved or transmitted with form descriptions.

8.26 itemlast

Identifies the last item on the page, excluding the global item. An item is *last* when it appears last in the build order (in other words, it is last in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<itemlast>item reference</itemlast>
```

item reference	*string*	a reference to the last item on the page, excluding the global item.

Available In

page global

Example

This sample shows how the *itemlast* option appears in the page globals of a form.

```
<page sid="PAGE1">
   <global sid="global">
      <itemlast>BUTTON2</itemlast>
   </global>
</page>
```

Usage Notes

1. Defaults: **none**
2. The *itemlast* option is not saved or transmitted with form descriptions.

8.27 itemlocation

Serves three purposes:

- It specifies the location of an item in the page layout.
- If the extent setting is used, it will set the size of an item's bounding box.
- If the relational positioning scheme is used, it may dynamically alter the size of an item.

Each parameter in the *itemlocation* option defines one aspect of an item's location or size.

There are two different schemes that can be used to position items on the page: absolute positioning and relative positioning. The absolute positioning scheme anchors the top left corner of an item to a particular pixel on the displayed page, whereas the relative positioning scheme places items on the page in relation to one another. You can combine the two schemes in the same form to gain the advantages of both.

For more information on the two schemes, see "Absolute Positioning Scheme" and "Relative Positioning Scheme" below.

Note: *The two methods of positioning can be combined so that some items are positioned absolutely, and other items are positioned in relation to those absolute items.*

Syntax

```
<itemlocation>
    <ae>
        specification_1
    </ae>
    ...
    <ae>
        specification_n
    </ae>
</itemlocation>
```

where: (absolute positioning and extent modifier)

- specification is defined as: modifier, x-coordinate, y-coordinate (relational positioning)
- specification is defined as: modifier, item tag_1, item tag_2

modifier	*(see below)*	■ the type of modification to apply to the item's location or size
x-coordinate	*short (must be positive if modifier is absolute)*	■ the horizontal distance in pixels from the form's top left corner (with the modifier absolute); or ■ the horizontal distance in pixels from the item's top left corner in its original position to its new offset position (with the modifier offset)
y-coordinate	*short (must be positive if modifier is absolute)*	■ the vertical distance in pixels from the form's top left corner (with the modifier absolute); or ■ the vertical distance in pixels from the item's top left corner in its original position and to its new offset position (with the modifier offset)
item tag	*string*	■ identifies the reference point item

Available In

box, button, check, field, label, line, list, popup, combobox, radio, spacer

Modifiers

There are four types of modifiers:

- position modifiers — used to position an item

- alignment modifiers — used to align one edge of an item (relative positioning only)
- expansion modifiers — used to alter an item's size (relative positioning only)
- the *extent* modifier — used to set a pixel based size for an item (both relative and absolute positioning)

Position Modifiers

For the Absolute Positioning Scheme:

Modifier	Description
absolute	places top left corner of item on the pixel noted in the x-coordinate and y-coordinate settings

The *extent* modifier, listed later in this section, can also be used with absolute positioning.

For the Relational Positioning Scheme

Modifier	Description
above	places item a small distance above reference point item; aligns left edges
after	places item a small distance after reference point item; aligns top edges
before	places item a small distance before reference point item; aligns top edges
below	places item a small distance below reference point item; aligns left edges
offset	places item so that it is offset from its original location by the measurement specified in the x-coordinate and y-coordinate settings
within	assigns item to the toolbar

Note: *If a specification contains the* within *modifier, it must be the first specification in the list.*

Alignment Modifiers (Relational Positioning only)

Modifier	Description
alignb2b	aligns bottom edge of item with bottom edge of reference point item
alignb2c	aligns bottom edge of item with vertical center of reference point item
alignb2t	aligns bottom edge of item with top edge of reference point item
alignc2b	aligns vertical center of item with bottom edge of reference point item
alignc2l	aligns horizontal center of item with left edge of reference point item
alignc2r	aligns horizontal center of item with right edge of reference point item
alignc2t	aligns vertical center of item with top edge of reference point item
alignhoriz between	aligns horizontal center of item between right edge of first reference point item and left edge of second reference point item
alignhorizc2c	aligns horizontal center of item with horizontal center of reference point item; center below
alignl2c	aligns left edge of item with horizontal center of reference point item
alignl2l	aligns left edge of item with left edge of reference point item
alignl2r	aligns left edge of item with right edge of reference point item
alignr2c	aligns right edge of item with horizontal center of reference point item
alignr2l	aligns right edge of item with left edge of reference point item
alignr2r	aligns right edge of item with right edge of reference point item
alignt2b	aligns top edge of item with bottom edge of reference point item
alignt2c	aligns top edge of item with vertical center of reference point item
alignt2t	aligns top edge of item with top edge of reference point item
alignvertbetween	aligns vertical center of item between bottom edge of first reference point item and top edge of second reference point item
alignvertc2c	aligns vertical center of item with vertical center of reference point item

Note: *The modifiers alignvertbetween and alignhorizbetween require two reference items.*

Expansion Modifiers (Relational Positioning only)

Modifier	Description
alignb2b	aligns bottom edge of item with bottom edge of reference point item
expandb2c	holds top edge of item constant and expands bottom edge to align with vertical center of reference point item
expandb2t	holds top edge of item constant and expands bottom edge to align with top edge of reference point item
expandl2c	holds right edge of item constant and expands left edge to align with horizontal center of reference point item
expandl2l	holds right edge of item constant and expands left edge to align with left edge of reference point item
expandl2r	holds right edge of item constant and expands left edge to align with right edge of reference point item
expandr2c	holds left edge of item constant and expands right edge to align with horizontal center of reference point item
expandr2l	holds left edge of item constant and expands right edge to align with left edge of reference point item
expandr2r	holds left edge of item constant and expands right edge to align with right edge of reference point item
expandt2b	holds bottom edge of item constant and expands top edge to align with bottom edge of reference point item
expandt2c	holds bottom edge of item constant and expands top edge to align with vertical center of reference point item
expandt2t	holds bottom edge of item constant and expands top edge to align with top edge of reference point item

The Extent Modifier (Relational and Absolute Positioning)

Modifier	Description
extent	holds the top left corner of the item in place, and sizes the item so that it is as many pixels wide as the x coordinate, and as many pixels tall as the y coordinate

Absolute Positioning Scheme

This scheme anchors an item to a particular coordinate on the visible page. The coordinate is a measurement in pixels of the distance between the top left corner of the form and the item's top left corner. This allows precise item locations that do not rely on the positioning of any other items. Absolutely positioned items contain all of their own positioning information.

Caution

There are three methods for sizing items when using absolute positoning: in pixels, characters, or inches. If you choose to size items using characters, you should be aware that different platforms and video cards use differently sized fonts. Even with so-called cross-platform fonts, an item's actual size (in pixels) might change from one platform to the next if it is sized using character height and width. As a result, absolutely positioned items sized with characters may have overlap or sizing problems if displayed on different platforms, different video cards, or in both small font and large font modes.

To ensure that forms appear the same on any platform, and under any video card or font mode while using absolute positioning, size items in pixels or inches.

Relative Positioning Scheme

Relative positioning allows an item to be placed relative to the location of another item. It also allows for the specification of an item's size relative to the size and location of other items. The other items (called reference points or anchor items) must be defined before they can be used in an *itemlocation* statement.

When using the relative positioning scheme, the first external item placed on the form appears in the top left corner. It cannot be placed in relation to any other item, since no other items exist. All subsequent items can be placed in relation to items that appear before them in the form's description. If no relational position for an item is specified, it will appear below the previous item, with its left edge against the page's left edge.

The relative positioning scheme also allows an item to be offset from its original position, by a particular number of pixels. This is a quick way to create an indented layout on a form.

It is valid to offset an item in any of these four directions: right, left, up, down. Since the offset is measured by a pixel grid and is represented with x and y coordinates, the directions left and up are measured as negative distances. For example, if you had a field aligned directly below an item, but wanted to offset it so that its left edge is further left than the label's, the x measurement would be negative, as in -15.

An item can be offset from its original relational position. For example,

```
<itemlocation>
   <ae>
      <ae>below</ae>
      <ae>persInfo_label</ae>
   </ae>
   <ae>
```

```
        <ae>offset</ae>
        <ae>-15</ae>
        <ae>20</ae>
    </ae>
</itemlocation>
```

Itemlocation can only reference items on the same page as the item being defined. If the item being defined is in a toolbar, the referenced items must be in the same toolbar.

The Extent Modifier

The extent modifier allows an absolute size for an item to be set in pixels. When an extent is specified, the item's top left corner will stay where it is, and the item will be resized so that it is as many pixels wide as the x value and as many pixels in height as the y value.

Note: *Itemlocation uses the bounding boxes of the defined and referenced items to determine location and size.*

Examples

Absolute Positioning

This sample places a label on the page so that its top left corner is 20 pixels in from the page's left edge, and 30 pixels down from the top of the page.

```
<label sid="persInfo_label">
    <value>Personal Information</value>
    <itemlocation>
        <ae>
            <ae>absolute</ae>
            <ae>20</ae>
            <ae>30</ae>
        </ae>
    </itemlocation>
</label>
```

Relative positioning

This sample aligns the vertical center of an item between the bottom edge of the item label_one and the top edge of the item label_two.

```
<itemlocation>
    <ae>
        <ae>alignvertbetween</ae>
        <ae>label_one</ae>
        <ae>label_two</ae>
```

```
    </ae>
</itemlocation>
```

This sample aligns the item's left edge with the center of item the_firm and expands the right edge to align with the right edge of the same reference item (the_firm).

```
<itemlocation>
    <ae>
        <ae>alignl2c</ae>
        <ae>the_firm</ae>
    </ae>
    <ae>
        <ae>expandr2r</ae>
        <ae>the_firm</ae>
    </ae>
</itemlocation>
```

This sample assigns an item to the toolbar main_toolbar and positions it under the company logo company_logo.

```
<itemlocation>
    <ae>
        <ae>within</ae>
        <ae>main_toolbar</ae>
    </ae>
    <ae>
        <ae>below</ae>
        <ae>company_logo</ae>
    </ae>
</itemlocation>
```

Extent

This sample shows an extent setting on a field that has been placed using absolute positioning. The field is first placed at an x-y coordinate of 10, 10. It is then sized to be 300 pixels wide and 30 pixels high.

```
<itemlocation>
    <ae>
        <ae>absolute</ae>
        <ae>10</ae>
        <ae>10</ae>
    </ae>
    <ae>
        <ae>extent</ae>
        <ae>300</ae>
        <ae>30<ae>
    </ae>
</itemlocation>
```

Usage Notes

1. Default item location:
   - in the body of the page
   - under the previous item in the page definition
   - aligned along the left margin of the page
2. Itemlocation overrides size. If the itemlocation (extent) affects the size of the item and the size option has also been set for the item, the itemlocation will determine the size.
3. An item's vertical center is halfway between the top and bottom edges. The horizontal center is halfway between the left and right edges.
4. To offset an item by shifting it to the right or down the page, specify the offset distance using positive integers. To offset an item by shifting it to the left or up the page, specify the offset distance using negative integers.

8.28 itemnext

Identifies the next item on the page, excluding the global item. An item is *next* when it appears next in the build order (in other words, it is next in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<itemnext>item reference</itemnext>
```

item reference	*string*	a reference to the next item on the page, excluding the global item.

Available In

action, box, button, cell, check, combobox, data, field, help, label, line, list, popup, radio, signature, spacer, toolbar

Example

This example shows the *itemnext* option for a label.

```
<label sid="LABEL2">
   <value>This is a label.</value>
   <itemnext>LABEL3</itemnext>
</label>
```

Usage Notes

1. Defaults: **none**
2. If the *itemnext* option is in the last item on the page, it points to the first item on the page (excluding the global item).
3. The *itemnext* option is not saved or transmitted with form descriptions.

8.29 itemprevious

Identifies the next item on the page, excluding the global item. An item is *previous* when it immediately precedes the current item in the build order (in other words, it is immediately previous in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<itemprevious>item reference</itemprevious>
```

item reference	*string*	a reference to the previous item on the page, excluding the global item.

Available In

action, box, button, cell, check, combobox, data, field, help, label, line, list, popup, radio, signature, spacer, toolbar

Example

This example shows the *itemprevious* option for a label.

```
<label sid="LABEL3">
   <value>This is a label.</value>
   <itemprevious>LABEL2</itemprevious>
</label>
```

Usage Notes

1. Defaults: **none**
2. If the *itemprevious* option is in the first item on the page (excluding the global item), it points to the last item on the page.
3. The *itemprevious* option is not saved or transmitted with form descriptions.

8.30 justify

Aligns lines of text within the space an item occupies.

Syntax

```
<justify>alignment</justify>
```

alignment	*left*	align each line's left edge along the left margin
	right	align each line's right edge along the right margin
	center	align the center of each line with the center of the item

Available In

button, combobox, field, label, popup

Example

This sample aligns the text in the center of the item.

```
<justify>center</justify>
```

Usage Notes

1. Default: varies depending on the item
   - *button*, *combobox* and *popup* items: **center**
   - *label* and *field* items: **left**
2. The built-in labels (*label* option) for items do not support a *justification* option.

8.31 keypress

Contains the last keystroke made by the user in the focused item, page, or form. A keypress option is ignored if no keypress has been established at the level of focus. If the value of a keypress option is ignored at the item level, it passes up to the page level, and if ignored at the page level, it passes up to the form level. This option allows for the creation of a default button (shortcut key) on a page or a form.

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<keypress>key pressed</keypress>
```

Key pressed	*ESC*	escape key
	ENTER	enter key
	NUMPAD ENTER	number pad enter key
	“ “	space bar
	F[1-12]	function key (can be any key from F1 to F12)
	any keyboard char	any key on the keyboard

Available In

button, field, combobox, popup, list, radio, check, page global, form global

Example

The following example shows how you can set the keypress option to respond to the F7 key. When the form’s user presses the F7 key, it activates the CancelButton’s cancel action and cancels the form.

```
<button sid="CancelButton">
   <type>cancel</type>
   <value>Cancel</value>
   <opt_1 compute="toggle(global.global.keypress, ’F7’) == &#xA;
      ’1’ ? set (’CancelButton.activated’,’on’) : ’’"></opt_1>
</button>
```

Usage Notes

1. Default: **none**
2. The *keypress* option enables the use of a default button or action on the form.
3. You cannot place a compute in the activated option of a button. You must place it in a custom option.
4. The *keypress* option is not saved or transmitted with form descriptions.

8.32 label

Specifies an external text label for an item. The label appears above the item and aligned with its left margin. The only exception is *popup* items, where the label appears inside the item.

Syntax

```
<label>label text</label>
```

label text	*string*	the text of the label

Available In

cell, check, combobox, field, list, popup, radio, spacer, page global, form global

Example

This sample defines a typical label.

```
<label>Student Registration Form</label>
```

Usage Notes

1. Default: **none**
2. The label defined in a *label* option has a transparent background by default (if no color is specified, the label background color will be the same as the page background color). To display a particular color behind the label, set the *labelbgcolor* option.
3. If used in the page global, the *label* option sets the title of the page as it appears in the title bar of the window displaying the form. If used in the form global, the *label* option sets the default title for all pages.
4. Multiple line labels require a carriage return in the code where you want it to appear in the label. For example:

   ```
   <label>This label spans
   two lines.</label>
   ```

8.33 labelbgcolor

Defines the background color for the label specified in the label option.

Syntax

```
<labelbgcolor>
   <ae>color name</ae>
</labelbgcolor>
```

or

```
<labelbgcolor>
   RGB triplet
</labelbgcolor>
```

Note:

- Either format is acceptable.

color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See “Data Type Designators Used in Option Descriptions” on page 85 for the syntax of an RGB triplet.

Available In

check, combobox, field, list, radio, page global, form global

Examples

These samples both set the background color to red.

```
<labelbgcolor>
   <ae>red</ae>
</labelbgcolor>

<labelbgcolor>
   <ae>255</ae>
   <ae>0</ae>
   <ae>0</ae>
```

```
</labelbgcolor>
```

Usage Notes

1. Default: **transparent**

   This means that a *label* option will always be transparent unless a color is specified (if no color is specified, the label background color will be the same as the page background color or an underlying box, if there is one).

8.34 labelborderwidth

Defines whether there is a border around the label specified in the *label* option.

Syntax

```
<labelborderwidth>width</labelborderwidth>
```

width	*numeric boolean*	**zero** for no border or **one** for a border

Available In

check, combobox, field, list, radio

Example

This sample sets the the label to have a border:

```
<labelborderwidth>1</labelborderwidth>
```

Usage Notes

1. Default: **zero**
2. The border is always one pixel in width.

8.35 labelfontcolor

Defines the font color for the label specified in the *label* option.

Syntax

```
<labelfontcolor>
   <ae>color name</ae>
</labelfontcolor>
```

or

```
<labelfontcolor>
   RGB triplet
</labelfontcolor>
```

Note:

- Either format is acceptable.

color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See “Data Type Designators Used in Option Descriptions” on page 85 for the syntax of an RGB triplet.

Available In

check, combobox, field, list, radio, page global, form global

Examples

These samples both set the font color to green1.

```
<labelfontcolor>
   <ae>green1</ae>
</labelfontcolor>

<labelfontcolor>
   <ae>0</ae>
   <ae>255</ae>
   <ae>0</ae>
</labelfontcolor>
```

Usage Notes

1. The default labelfontcolor is the page fontcolor setting or default. If there is no fontcolor setting at the page level, the page fontcolor is derived from the form fontcolor setting or default (black).

8.36 labelfontinfo

Defines the font name, point size, and font characteristics for the label specified in the label option.

Syntax

```
<labelfontinfo>
   <ae>font name</ae>
   <ae>point size</ae>
   <ae>weight</ae>
   <ae>effects</ae>
   <ae>form</ae>
   <ae>character set</ae>
</labelfontinfo>
```

Note:

- weight, effects, form, and character set are optional.

font name	*string*	the name of the font
point size	*unsigned byte*	the size of the font
weight	*plain*	use plain face
	bold	use bold face
effects	*underline*	underline the text
form	*italics*	use the italic form
character set	*string*	the name of the character set

Available In

check, combobox, field, list, radio, page global, form globals

Example

This sample sets the font information to Palatino 12, plain (the default), underlined in the ANSI character set.

```
<labelfontinfo>
   <ae>Palatino</ae>
   <ae>12</ae>
```

```
   <ae>underline</ae>
</labelfontinfo>
```

Usage Notes

1. Defaults:
   - font name: **Helvetica**
   - point size: **8**
   - weight: **plain**
   - effects: **not underlined**
   - form: **not italics**
   - character set: **ANSI**
2. If any of the fontinfo settings are invalid, then the defaults will be used.
3. The *size* option calculates item size using the font's average character width. Therefore, choice of font affects item width.
4. XFDL supports the following fonts, font sizes, and character sets:
   - Fonts: Courier, Times, Symbol (symbol), Helvetica, and Palatino.
   - Sizes: 8, 9, 10, 11, 12, 14, 16, 18, 24, 36, 48.
   - Character Sets: ANSI and Symbol.
5. Other fonts and font sizes may be used. However, especially for cross-platform Internet applications, it is best to choose from the ones cited above since they are guaranteed to work.

8.37 layoutinfo

This option records location information for all visible signed items. A hash is taken of each page containing a signed item, and this hash includes positioning information for all the signed items relative to each other in those pages.

Syntax

```
<layoutinfo>
    <pagehashes>
        <PAGE1>pagehash1</PAGE1>
        <PAGEn>pagehashn</PAGEn>
    </pagehashes>
</layoutinfo>
```

pagehashes	*string*	stores page hashes for pages containing signed items

Available In

signature

Example

In the following example, the signature item includes hashes for the three pages on which items were signed:

```
<signature sid="SIGNATURE1">
    <layoutinfo>
        <pagehashes>
            <PAGE1>pJAdX7+zh9+zEe/ofVskig==</PAGE1>
            <PAGE2>kyLkNpOlH2WBY6RsPGcQIg==</PAGE2>
            <PAGE3>3vCu9IIu/5QFivDD7UvOAg==</PAGE3>
        </pagehashes>
    </layoutinfo>
</signature>
```

Usage Notes

1. This option helps ensure the security of your forms when you have signed items that are positioned using relative positioning, and unsigned *reference items*. Reference items are items on which the relative positioning of other items is based. Without layoutinfo, it is possible to move signed items on the form — without breaking the signature — by moving their unsigned reference items. By

using layoutinfo to record the position of the signed items relative to each other, you can detect this form of tampering.

8.38 linespacing

This option adjusts the spacing between lines of text. This sets on offset value, which will add to or subtract from the default spacing. For example, a value of 1 will add one pixel to the space between each line, while a value of -1 will remove one pixel from the space between each line.

Syntax

```
<linespacing>offset</linespacing>
```

offset	*integer*	the offset measured in pixels. This can be a positive or negative integer, or a value of 0 for no offset.

Available In

button, label, spacer

Example

In the following example, the linespacing for a label is set to 12, which will create a large amount of space between lines:

```
<label sid="LABEL1">
   <size>
      <ae>10</ae>
      <ae>5</ae>
   </size>
<value>
```

This label will have a lot of space between lines:

```
</value>
   <linespacing>12</linespacing>
</label>
```

Usage Notes

1. When using a negative offset, the offset cannot be larger than the size of the font (that is, the second line of text cannot be moved higher than the first line). For example, if the font was 12 pixels high, the offset could not be larger than -12.
2. When using a positive offset, there is no limit to the size of the offset.
3. For buttons and labels, the linespacing affects the text in the *value* option. For spacers, the linespacing affects the text in the *label* option.

8.39 mimedata

Contains the actual data associated with a *data* item or a *signature* item. It can be binary data or the contents of an enclosed file. The data is encoded in base64 format, so that even forms containing binary data can be viewed in a text editor. When the data is needed by the form, it is decoded automatically from base64 back to its native format.

Data may also be compressed before base64 encoding, allowing an item to store a larger block of data.

About MIME data in signature items

The MIME data contains the contents of a signature. An XFDL generator must create it as follows:

1. Using the signature filter instructions in the associated signature button, create a plain-text version of the form or portion of the form to be signed.
2. Using the instructions in the signature button's *signformat* option, create a hash of the plain-text description.
3. Sign the hash with the signer's private key.
4. Include a binary represenation of the signature (as generated by the signature engine) in the *mimedata* option.

Syntax

```
<mimedata encoding="format">MIME data</mimedata>
```

format	*string*	the format in which to encode the data. Valid formats are: ■ base64 — a base 64 textual representation of the data. ■ base64-gzip — a base 64 textual represenation of the data after it has been gzip compressed.
MIME data	*string*	the binary data or enclosed file contents

Available In

data, signature

Example

This sample assigns some encoded data to the mimedata option:

```
<mimedata encoding="base64-gzip">
```

```
    R0lGODdhYABPAPAAAP///wAAACwAAAAAYABPAAAC/4SPqcvtD02Y
    Art68+Y7im7ku2KkzXnOzh9v7qNw+k+TbDoLFTvCSPzMrS2YzmTE
</mimedata>
```

This sample shows a mimedata option in a digital signature:

```
<signature sid="empSignature">
    <signformat>application/vnd.xfdl</signformat>
    <signer>Jane D Smith, jsmith@insurance.com</signer>
    <signature>Page1.empSignature</signature>
    <signitemrefs>
        <ae>omit</ae>
        <ae>Page1.mgrSigButton</ae>
        <ae>Page1.admSigButton</ae>
        <ae>Page1.empSignature</ae>
        <ae>Page1.mgrSignature</ae>
        <ae>Page1.admSignature</ae>
    </signitemrefs>

<!-- The items listed above MUST have itemlocation
options with absolute and extent as the last
settings in order for the filter below to
be sufficient in terms of security -->

    <signoptionrefs>
        <ae>keep</ae>
        <ae>PAGE1.mgrSigButton.itemlocation</ae>
        <ae>PAGE1.admSigButton.itemlocation</ae>
        <ae>PAGE1.empSignature.itemlocation</ae>
        <ae>PAGE1.mgrSignature.itemlocation</ae>
        <ae>PAGE1.admSignature.itemlocation</ae>
    </signoptionrefs>

    <signoptions>
        <ae>omit</ae>
        <ae>triggeritem</ae>
        <ae>coordinates</ae>
    </signoptions>
    <mimedata encoding="base64">
        MIIFMgYJKoZIhvcNACooIIFIzCCBR8CAQExDzANBgkgAQUFADA
        LB\ngkqhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl6hMrc5D
        ySSP+X5j\nANfBGSOI\n9w0BAQQwDwYDVQQHEwhJbmRlcm5lqw
        dDEXMBUGA1UEChM\nOVmVyaVNpZ24sIEluYy4xNDAKnqweaftn
        1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmRdWFsIFN1YnNjcml
        iyZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCAExETA
    </mimedata>
</signature>
```

Usage Notes

1. Default: **none**
2. Base64 encoding transforms the data into a format that can be processed easily by text editors, email applications, and so on. Converting data to base64 format ensures the resulting string contains no characters requiring an escape sequence.
3. Base64-gzip encoding compresses the data before transforming it to base64 format.

Note: *For signatures: because the signer's public key is included in the MIME data, a subsequent program can verify a signature without requiring that the signer's key be previously installed.*

8.40 mimetype

Defines the MIME type of the data stored in a *data* item.

Syntax

```
<mimetype>MIME type</mimetype>
```

MIME type	*string*	the MIME type of the data item

Available In

data

Example

This sample sets the MIME type to indicate image data:

```
<mimetype>image/gif</mimetype>
```

Usage Notes

1. Default: **application/vnd.xfdl**
2. Here are some examples of MIME types. For full information on MIME types, read the MIME rfcs (1521, 1522, 1867 and 822). They are available on the World Wide Web.

MIME type	Description
application/vnd.xfdl	XFDL form item
image/jpeg	image item
image/rast	image item
image/tiff	image item
image/png	image item
image/bmp	image item
text/plain	ASCII text item

8.41 mouseover

Specifies whether the mouse pointer is currently over an item or page.

This option is set by an external program such as a parser, and is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<mouseover>status</mouseover>
```

status	*on*	mouse pointer is over item or page
	off	mouse pointer is not over item or page

Available In

button, check, combobox, field, list, popup, radio, toolbar, page global

Example

The following example shows a button that changes its color to white if it the mouse pointer is over it, and to blue if the pointer is not over it.

```
<button sid="saveButton">
   <type>save</type>
   <value>Save</value>
   <bgcolor>
      <ae compute="mouseover == 'on' ? 'white' : 'blue'"></ae>
   </bgcolor>
</button>
```

Usage Notes

1. Default: **off**
2. An object's *mouseover* option is set to **on** when the mouse pointer is over the object, and to **off** when the mouse pointer is not over the object.
3. A page global *mouseover* option is set to **on** when the mouse pointer is over the page (even if it is also over an item on the page).
4. A *mouseover* option in a toolbar is set to **on** when the mouse pointer is over the toolbar (even if it is also over an item in the toolbar).

5. The *mouseover* option is not included in form descriptions that are saved or transmitted.

8.42 next

Identifies the item to receive focus when a user tabs ahead from the current item. If a user tabs ahead from the last item on the page, the tab cycles within the same page, beginning with the first item on the page. Only modifiable or readonly items can receive focus.

Syntax

```
<next>item reference</next>
```

item reference	*string*	identifies the item to receive focus next

Available In

button, check, combobox, field, list, popup, radio, page global

Example

This sample points to the item “address_field”. When users tab ahead from the current item, the item identified as “address_field” will receive focus.

```
<next>address_field</next>
```

Usage Notes

1. Default: **none**
2. The first page defined in the form is always the first page displayed. The default tabbing order depends on the order in which page and item definitions occur within the form definition. The sequence is as follows:
   - First item to receive focus: first modifiable item defined for the body of the first page
   - Subsequent items to receive focus: each modifiable item on the page in the order in which they are defined
3. If the last item on the page is tabbed past, the first modifiable item in the page’s toolbar receives focus. If there is no toolbar, focus returns to the first item.
4. Placing next in page globals defines the first item to receive focus when the page appears.
5. If the *next* option identifies page globals, focus moves to the item defined to receive focus when the page appears. The page globals reference is *global* for the current page or *page_tag.global* for another page.

8.43 pagefirst

Identifies the first page of the form, excluding the global page. A page is *first* when it appears first in the build order (in other words, it is first in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<pagefirst>page reference</pagefirst>
```

page reference	*string*	a reference to the first page in the form, excluding the global page.

Available In

page global

Example

This sample shows how the *pagefirst* option appears in the page globals of a form.

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

Usage Notes

1. Defaults: **none**
2. The *pagefirst* option is not saved or transmitted with form descriptions.

8.44 pageid

Defines a unique identifier for the page.

Syntax

```
<pageid>
   <serialnumber>serial number</serialnumber>
</pageid>
```

serial number	*string*	provides a unique identifying string for the form. This is generated by an external program such as a form design program.

Available In

page global

Example

This sample shows how the *pageid* option appears in the page globals of a form.

```
<pageid>
   <serialnumber>{94EC2BA4-7D34-B5E4-0060-9947}</serialnumber>
</pageid>
```

Usage Notes

1. Defaults: **none**
2. This option is intended for use with form design programs and licensing models.

8.45 pagelast

Identifies the last page of the form, excluding the global page. A page is *last* when it appears last in the build order (in other words, it is last in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<pagelast>page reference</pagelast>
```

page reference	*string*	a reference to the last page in the form, excluding the global page.

Available In

page global

Example

This sample shows how the *pagelast* option appears in the page globals of a form.

```
<page sid="PAGE1">
   <global sid="global">
      <pagelast>PAGE4</pagelast>
   </global>
</page>
```

Usage Notes

1. Defaults: **none**
2. The *pagelast* option is not saved or transmitted with form descriptions.

8.46 pagenext

Identifies the next page in the form, excluding the global page. A page is *next* when it appears next in the build order (in other words, it is next in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<pagenext>page reference</pagenext>
```

page reference	*string*	a reference to the next page in the form, excluding the global page.

Available In

page global

Example

This example shows the *pagenext* option for PAGE2 of a form.

```
<page sid="PAGE2">
   <global sid="global">
      <pagenext>PAGE3</pagenext>
   </global>
</page>
```

Usage Notes

1. Defaults: **none**
2. If the *pagenext* option is on the last page of the form, it points to the first page of the form (excluding the global page).
3. The *pagenext* option is not saved or transmitted with form descriptions.

8.47 pageprevious

Identifies the previous page in the form, excluding the global page. A page is *previous* when it immediately precedes the current page in the build order (in other words, it is immediately previous in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

Syntax

```
<pageprevious>page reference</pageprevious>
```

page reference	*string*	a reference to the previous page in the form, excluding the global page.

Available In

page global

Example

This example shows the *pageprevious* option for PAGE2 of a form.

```
<page sid="PAGE2">
   <global sid="global">
      <pageprevious>PAGE1</pageprevious>
   </global>
</page>
```

Usage Notes

1. Defaults: **none**
2. If the *pageprevious* option is on the first page of the form (excluding the global page), it points to the last page of the form.
3. The *pageprevious* option is not saved or transmitted with form descriptions.

8.48 previous

Identifies the item to receive focus when a user tabs backwards, using SHIFT + TAB, from the current item. If the current item has a *previous* option, the item indicated in that option is next in the reverse tab order. If the current item has no *previous* option, the previous item in the build order that can receive the input focus is next in the reverse tab order.

Syntax

```
<previous>item reference</previous>
```

item reference	*string*	identifies the item to receive focus next

Available In

button, check, combobox, field, list, popup, radio, page global

Example

This sample points to the item “date_field”. When users tab back from the current item, the item identified as “date_field” will receive focus.

```
<previous>date_field</previous>
```

Usage Notes

1. The first page defined in the form is always the first page displayed. The default tabbing order depends on the order in which page and item definitions occur within the form definition. The sequence is as follows:
   - First item to receive focus: first modifiable item defined for the body of the first page
   - Subsequent items to receive focus: each modifiable item on the page in the reverse order in which they are defined
2. When tabbing back past the first item on the page, the last modifiable item in the page’s toolbar receives focus. If there is no toolbar, focus returns to the last item defined in the page.
3. Placing previous in page globals defines the first item to receive focus when the page appears.
4. If the *previous* option identifies page globals, focus moves to the item defined to receive focus when page appears. The page globals reference is *global* for the current page or *page_tag.global* for another page.

8.49 printbgcolor

Enables the form to be printed with a specific background color on a color printer. This color can be the same as or different from the background color shown on the screen. On black and white printers, grayscaling is used.

See also: printlabelbgcolor, printfontcolor, printlabelfontcolor.

Syntax

```
<printbgcolor>
   RGB triplet
</printbgcolor>
```

or

```
<printbgcolor>
   <ae>color name<ae>
</printbgcolor>
```

Note:

- Either format is acceptable.

RGB triplet	*color*	the RGB triplet. See “Data Type Designators Used in Option Descriptions” on page 85 for the syntax of an RGB triplet.
color name	*color*	the color name

Available In

box, button, check, combobox, field, label, list, popup, radio, page global, form global

Examples

These samples both set the printed background color to forest green.

```
<printbgcolor>
   <ae>forest green</ae>
</printbgcolor>
<printbgcolor>
   <ae>34</ae>
   <ae>139</ae>
   <ae>34</ae>
```

```
</printbgcolor>
```

Usage Notes

1. Defaults behave as follows:
   - Form: **white**
   - Page: the form global setting or default (white)
   - Item: when no printbgcolor is specified, the printbgcolor default is derived from the item's specified or default bgcolor (which may be further derived from the page or form global bgcolor settings or defaults).

8.50 printfontcolor

Enables the item to be printed with a specific font color on a color printer. This color can be the same as or different from the font color shown on the screen. On black and white printers, grayscaling is used.

Syntax

```
<printfontcolor>
   RGB triplet
</printfontcolor>
```

or

```
<printfontcolor>
   <ae>color name<ae>
</printfontcolor>
```

Note:

- Either format is acceptable.

RGB triplet	*color*	the RGB triplet. See “Data Type Designators Used in Option Descriptions” on page 85 for the syntax of an RGB triplet.
color name	*color*	the color name

Available In

button, check, combobox, field, label, line, list, popup, radio, page global, form global

Examples

These samples both set the printed font color to forest green.

```
<printfontcolor>
   <ae>forest green</ae>
</printfontcolor>

<printfontcolor>
   <ae>34</ae>
   <ae>139</ae>
   <ae>34</ae>
</printfontcolor>
```

Usage Notes

1. The default printfontcolor is derived from the fontcolor setting or default for the item. (The fontcolor default may be further derived from page or form settings or defaults.)
2. For *check* items, the *printfontcolor* option describes what color the check in the check box will be when the form is printed. Likewise, for *radio* items, the *printfontcolor* option describes what color the **on** dot will be when the form is printed.
3. For *line* items, the *printfontcolor* option describes what color a line will be when the form is printed.

8.51 printlabelbgcolor

Enables a item's built-in label to be printed with a specific background color on a color printer. This color can be the same as or different from the background color shown on the screen. On black and white printers, grayscaling is used.

Syntax

```
<printlabelbgcolor>
   RGB triplet
</printlabelbgcolor>
```

or

```
<printlabelbgcolor>
   <ae>color name<ae>
</printlabelbgcolor>
```

Note:

- Either format is acceptable.

RGB triplet	*color*	the RGB triplet. See "Data Type Designators Used in Option Descriptions" on page 85 for the syntax of an RGB triplet.
color name	*color*	the color name

Available In

check, combobox, field, list, radio

Examples

These samples both set the printed label background color to forest green.

```
<printlabelbgcolor>
   <ae>forest green</ae>
</printlabelbgcolor>

<printlabelbgcolor>
   <ae>34</ae>
   <ae>139</ae>
   <ae>34</ae>
```

```
</printlabelbgcolor>
```

Usage Notes

1. Default: the printlabelbgcolor default is derived from the item's specified or default labelbgcolor.

8.52 printlabelfontcolor

Enables an item's built-in label to be printed with a specific font color on a color printer. This color can be the same as or different from the font color shown on the screen. On black and white printers, grayscaling is used.

Syntax

```
<printlabelfontcolor>
   RGB triplet
</printlabelfontcolor>
```

or

```
<printlabelfontcolor>
   <ae>color name<ae>
</printlabelfontcolor>
```

Note:

- Either format is acceptable.

RGB triplet	*color*	the RGB triplet. See "Data Type Designators Used in Option Descriptions" on page 85 for the syntax of an RGB triplet.
color name	*color*	the color name

Available In

check, combobox, field, list, radio

Examples

These samples both set the printed label font color to forest green.

```
<printlabelfontcolor>
   <ae>forest green</ae>
</printlabelfontcolor>

<printlabelfontcolor>
   <ae>34</ae>
   <ae>139</ae>
   <ae>34</ae>
```

```
</printlabelfontcolor>
```

Usage Notes

1. Default: the printlabelfontcolor default is derived from the item's specified or default labelfontcolor.

8.53 printsettings

Determines the settings that will be used when the form is printed. The user can be allowed to change these defaults, or the form can be set so that it will always follow the defaults.

Syntax

```
<printsettings>
   <pages>page list</pages>
   <dialog>dialog settings</dialog>
   <borderwidth>border setting</borderwidth>
</printsettings>
```

Note:

- All settings are optional.

page list	*(see below)*	the list of pages that print
dialog settings	*(see below)*	determines whether the print dialog is shown, and which settings must be used when printing (for example, paper orientation and number of copies)
border setting	*integer*	prints borders around form pages; 0 is off, and 1 or above sets the bordwidth to that number of pixels; default is 0

Available In

action, button, cell, page global, form global

Page List

The page list uses the following syntax:

```
<pages>
   <ae>keep/omit</ae>
   <ae>page tag 1</ae>
   <ae>page tag 2</ae>
   ...
</pages>
```

The settings for the page list work as follows:

Setting	Description
keep	the pages listed will be printed. Any other pages will not.
omit	the pages listed will not be printed. Any other pages will.
page tag	the list of page tags indicates which pages will be either kept or omitted.

Dialog Settings

The dialog settings use the following syntax:

```
<dialog>
   <active>on/off</active>
   <copies>#</copies>
   <orientation>portrait/landscape</orientation>
   <printpages>print page settings</printpages>
</dialog>
```

The settings work as follows:

Setting	Description
active	when **on**, the print dialog will be displayed before the form is printed, allowing the user to change the settings. When **off**, the dialog will not be shown and the form will be printed immediately.
copies	a positive integer that determines the number of copies that will be printed; default is 1.
orientation	determines whether the form will be printed in **landscape** or **portrait** orientation.
printpage settings	(see below)

Print Page Settings

The print page settings use the following syntax:

```
<printpages>
   <active>on/off</active>
   <choice>all/current</choice>
</printpages>
```

The settings work as follows:

Setting	Description
active	when **on**, the print dialog will allow the user to choose between printing “All” pages or the “Current Page”; default is **off**.
choice	**all** sets the printpages default to “All”; **current** sets the printpages default to “Current Page”; default is **all**.

Example

This sample prevents “page2” from being printed, sets the form to print in **landscape** orientation, and causes two copies of the form to be printed. The user is able to change all of these settings.

```
<printsettings>
   <pages>
      <ae>omit</ae>
      <ae>page2</ae>
   </pages>
   <dialog>
      <active>on</active>
      <orientation>landscape</orientation>
      <copies>2</copies>
   </dialog>
</printsettings>
```

Usage Notes

1. Default Page List: the page list will default to keeping **all** pages in the form.
2. Default Dialog Settings: the dialog will default to being **on**, and will print one copy of all pages in the form in a portrait orientation. By default, the user will be able to change all of these settings.

8.54 rtf

Stores the rich text value for rich text fields.

Syntax

```
<rtf>rich text</rtf>
```

rich text	*string*	the rich text string.

Available In

field

Example

This sample shows a rich text field:

```
<field sid="richTextField">
   <texttype>text/rtf</texttype>
   <value>Hello</value>
   <rtf>rich text version of Hello</rtf>
</field>
```

Usage Notes

1. **Default**: if the *rtf* option is empty or does not exist, a default rich text string is created as follows:
   - Text: set to equal the *value* option.
   - Font: set as the *fontinfo* and *fontcolor* options.
   - Justification: set as the *justify* option.

   If the required options are not present, then the default settings for those options are used. Note that the background color is controlled by the *bgcolor* option, not the rich text.
2. To use rich text in a field, the *texttype* option must be set to *text/rtf*.
3. A plain text version of the rich text string is stored in the *value* option, with the following considerations:
   - If the *value* option has a compute, that compute is lost when the text is copied from the *rtf* option. (However, external changes to the *value* are still made. For example, a *set*.)

- Once the text is copied to the *value* option, the format system may impose a change based on the *format* option. If a change occurs, then the *rtf* option is reset based on the formatted *value* option and the *fontinfo*, *fontcolor*, *justify*, and *bgcolor* options.
- If the plain text version does not match the rich text, then the rich text takes precedence. In this case, the *value* option is updated to reflect the rich text.

4. Computed changes to the *fontinfo*, *fontcolor*, *bgcolor*, *justify*, *texttype*, or *value* options will cause the *rtf* option to update if necessary.
5. Changes to the *rtf* option do not cause changes to the *fontinfo*, *fontcolor*, *bgcolor*, or *justify* options.

8.55 saveformat

Specifies the format a form will be saved in. An XFDL form may be saved in XFDL format or HTML format. Furthermore, the XFDL format may be compressed using ASCII compression.

The formats work as follows:

- XFDL format saves the entire form definition, including the user input.
- HTML format saves the form as a series of assignment statements for each modifiable item, equating the item reference with the item's value. The only items included in the save are custom items and the following modifiable items: *check*, *field*, *list*, *popup*, *combobox* and *radio*.

Syntax

```
<saveformat>MIME type</saveformat>
```

MIME type	*application/vnd.xfdl*	use XFDL format
	application/vnd.xfdl;content-encoding="base64-gzip"	use compressed XFDL format
	application/x-www-form-urlencoded	use HTML format

Available In

button, cell, page global, form global

Examples

This example shows how to use saveformat in a save button:

```
<button sid="save_button">
   <type>save</type>
   <saveformat>application/x-www-form-urlencoded</saveformat>
</button>
```

When a user clicks this button, the form will be converted to HTML format (see Usage Note 3 below) and saved to the user's drive.

XFDL format in form globals

This example shows how to use saveformat as a form global characteristic.

```
<?xml version="1.0"?>
<XFDL xmlns="http://www.PureEdge.com/XFDL/6.0"
```

```
xmlns:xfdl="http://www.PureEdge.com/xfdl/6.0.0">
<globalpage sid="global">
   <global sid="global">
      <bgcolor>
         <ae>ivory</ae>
      </bgcolor>
      <saveformat>application/vnd.xfdl</saveformat>
   </global>
</globalpage>
<page sid="page_1">
   ...
```

Any time a user saves this form, it will be saved in XFDL format.

Usage Notes

1. Default: The default format is the format that the form was in before it was parsed. For example, a form written in XFDL will be transmitted in XFDL, unless otherwise specified by this option.
2. This option can also be included as a form global option and in the definitions of items that trigger save actions. These are button or cell items that have a *type* option setting of **save**.

HTML Format by Item Type

The general syntax of a form saved in HTML format is:

```
itemreference=value&item reference=value&...
```

Note:

- The ampersand separates form items.

The syntax of items saved in HTML format by type:

Item Type	HTML Format
check	item tag=value option setting
field	item tag=value option setting
list	item tag=value option setting of selected cell Note that the item reference identifies the list.

Item Type	HTML Format
popup	item tag= value option setting of selected cell Note that the item reference identifies the popup.
combobox	item tag= value option setting
radio	group option setting=item tag of selected radio
custom	item tag=value option setting

Substitutions and Omissions:

- Only modifiable items are saved as HTML data. A form cannot be saved in HTML format and expected to be viewed as a form again. It is saved as a string of item tags and their associated values.
- Spaces in the value are replaced by the plus sign (+). For example, ‘Two words’ becomes ‘Two+words’.
- The membership operator in item and group references is replaced by a minus sign.
- *page_one.age_group* becomes *page_one-age_group*.
- Page tags are removed from item and group references in single page forms.
- Check boxes and radio buttons with a value option setting of **off** are omitted.
- Entries resulting in an empty string on the right hand side of the assignment statement are omitted. This occurs when the referenced option setting is empty or the option definition is missing.

8.56 scrollhoriz

Defines horizontal scrolling options for a *field* item.

Syntax

```
<scrollhoriz>option</scrollhoriz>
```

option	*never*	permit scrolling using the cursor but display no horizontal scroll bar
	always	permit scrolling and display a horizontal scroll bar
	wordwrap	wrap field contents from line to line, inhibit scrolling and display no horizontal scroll bar

Available In

field

Example

This sample sets the horizontal scrolling option to permit scrolling and to display the horizontal scroll bar.

```
<scrollhoriz>always</scrollhoriz>
```

Usage Notes

1. Default: **never**
2. The scroll bar displays along the field's bottom edge.

8.57 scrollvert

Defines vertical scrolling options for a *field* item.

Syntax

```
<scrollvert>option</scrollvert>
```

option	*never*	permit scrolling using the cursor but display no vertical scroll bar
	always	permit scrolling and display a vertical scroll bar
	fixed	inhibit scrolling and display no vertical scroll bars

Available In

field

Example

This sample sets the vertical scrolling option to inhibit all scrolling.

```
<scrollvert>fixed</scrollvert>
```

Usage Notes

1. Default: **never**
2. The scroll bar displays along the field's right edge.

8.58 signature

Used in conjunction with the button item to establish the XFDL item name by which a particular signature will be identified.

Syntax

```
<signature>name of signature</signature>
```

name of signature	*string*	the name of the signature

Available In

button, signature

Example

This sample identifies the signature item for a particular button as “mysig”.

```
<signature>mysig</signature>
```

Usage Notes

1. Default: **none**
2. The signature option must be included in each signature button that is set up.

8.59 signatureimage

Points to a data item, identifying it as the data item into which the captured signature image is placed. Used only with image-based digital signatures (such as CIC InkTools).

Syntax

```
<signatureimage>data item</signatureimage>
```

data item	*string*	the itemref for the data item that contains an image associated with a signature

Available In

button

Example

This sample identifies the data item “SIGIMAGE”. When the user signs a form with a pen/pad device, the mimedata of the captured image is placed in the data item “SIGIMAGE”.

```
<signatureimage>SIGIMAGE</signatureimage>
```

Usage Notes

1. Default: **none**
2. This option is used with signature types that utilize a digital image, such as InkTools signatures.

8.60 signdatagroups

Specifies which datagroups are filtered for a particular signature. Filtering a datagroup means keeping or omitting all of the *data* items that are in the specified datagroup.

For example, if a *signdatagroups* option specifies that the “attachments” datagroup should be kept, then all *data* items within the “attachments” group will be signed.

This filter applies to all *data* items present at the time of signing, including those added as enclosures.

Syntax

```
<signdatagroups>
   <ae>datagroup filter</ae>
   <ae>datagroup reference1</ae>
   ...
   <ae>datagroup referencen</ae>
</signdatagroups>
```

Note:

- There may be any number of *datagroup reference* entries.

datagroup filter	*keep*	include datagroups in the *datagroup reference* list with the signature; omit those not in the list
	omit	omit datagroups in the *datagroup reference* list from the signature; include those not in the list
datagroup reference	*string*	identifies a datagroup whose *data* items will be filtered

Available In

button, signature

Example

This example specifies a *signdatagroups* option that keeps the datagroup called “Business_Letters”.

```
<signdatagoups>
   <ae>keep</ae>
   <ae>Business_Letters</ae>
</signdatagroups>
```

Usage Notes

1. Default: **omit nothing (keep all data items), unless the containing page is omitted**
2. Since enclosed files can belong to several datagroups, and datagroups can contain several enclosed files, care must be exercised when setting up *signdatagroups* options to ensure that only the desired datagroups are filtered.
3. Other filters may take precedence over the *signdatagroups* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.61 signdetails

Specifies which certificate attributes are shown to the user when they are choosing a certificate to sign the form, and defines the filters used to select the available certificates when the user is signing a form.

For example, the *signdetails* option could specify that only those certificates with a common name that begins with “Bob” are shown, and that only the owner’s common name and email address are shown.

Syntax

```
<signdetails>
   dialogcolumns
   filteridentity
</signdetails>
```

Note:

- Both *dialogcolumns* and *filteridentity* are optional.

Expression	Setting	Description
dialogcolumns	(see below)	a list of certificate attributes that should be shown to the user when they are selecting a certificate to sign
filteridentity	(see below)	a list of certificate attributes and values that are used to filter which certificates are available to the user for signing

dialogcolumns

The *dialogcolumns* element uses the following syntax:

```
<dialogcolumns>
   <ae>attribute_1</ae>
   ...
   <ae>attribute_n</ae>
</dialogcolumns>
```

Note:

- The number of attributes is optional.

Each certificate attribute listed is shown to the user when they view the certificates available for signing. For example, if you wanted the user to see the owner's common name and email address for each certificate, you would use the following setting:

```
<dialogcolumns>
   <ae>Subject: CN</ae>
   <ae>Subject: E</ae>
</dialogcolumns>
```

For a list of available attributes, see "Certificate Attributes" below.

filteridentity

The *filteridentity* element uses the following syntax:

```
<filteridentity>
   <ae content = "array">
      <tag>attribute1</tag>
      <value>filter1</value>
   </ae>
   ...
   <ae>
      <tag>attributen</tag>
      <value>filtern</value>
   </ae>
</filteridentity>
```

Note:

- The number of attributes and filters is optional.

attribute	*string*	the name of the attribute you want to user to filter the available certificates
filter	*string*	the value to which you want to compare the attribute. Use an asterisk (*) as a wildcard or multiple characters, or a question mark (?) as a wildcard for a single character.

If the value of the attribute matches the filter, then the certificate will be available to the user. For example, to restrict the available certificates to those with a common name beginning with “Bob”, you would use the following filter:

```
<filteridentity>
   <ae>
      <tag>Subject: CN</tag>
      <value>Bob*</value>
   </ae>
</filteridentity>
```

For a list of available attributes, see “Certificate Attributes” below.

Available In

button, signature

Example

This example specifies a *signdetails* option that makes those certificates with an email address in the pureedge domain available, and shows the serial number and the owner's common name for each certificate.

```
<signdetails>
   <dialogcolumns>
      <ae>Serial</ae>
      <ae>Subject: CN</ae>
   </dialogcolumns>
   <filteridentity>
      <ae>
         <tag>Subject: E</ae>
         <value>*@pureedge.com</value>
      </ae>
   </filteridentity>
</signdetails>
```

Usage Notes

1. Default: all certificates are available, and the certificate's common name and expiry date are shown to the user.

Certificate Attributes

The following is a list of attributes that are common to X.509 certificates. Note that the names of certificate attributes are case sensitive.

Attribute	Description
Version	the version of the X.509 specification that the certificate follows
Serial	the certificate's serial number
SignatureAlg	the algorithm used by the Certificate Authority to sign the certificate
BeginDate	the date at which the certificate became valid
EndDate	the certificate's expiry date
PublicKey	the certificate's public key
FriendlyName	the certificate's friendly name
Subject: CN	the certificate owner's common name
Subject: E	the certificate owner's email address

Attribute	Description
Subject: T	the certificate owner's title
Subject: L	the certificate owner's locality
Subject: ST	the certificate owner's state of residence
Subject: O	the organization to which the certificate owner belongs
Subject: OU	the name of the organizational unit to which the certificate owner belongs
Subject: C	the certificate owner's country of residence
Subject: STREET	the certificate owner's street address
Subject: ALL	the certificate owner's complete distinguished name
Issuer: CN	the certificate issuer's common name
Issuer: E	the certificate issuer's email address
Issuer: T	the certificate issuer's title
Issuer: L	the certificate issuer's locality
Issuer: ST	the certificate issuer's state of residence
Issuer: O	the organization to which the certificate issuer belongs
Issuer: OU	the organizational unit to which the certificate issuer belongs
Issuer: C	the certificate issuer's country of residence
Issuer: STREET	the certificate issuer's street address
Issuer: ALL	the certificate issuer's complete distinguished name

8.62 signer

Identifies who signed a particular form.

Syntax

```
<signer>identity of user</signer>
```

Identity of user	*string*	identity of user

Available In

button, signature

Example

In this example, signer is similar to a user's email signature, clearly identifying who signed the form.

```
<signer>John Smith, jsmith@acme.org</signer>
```

Usage Notes

1. The setting of the *signer* option varies, depending on the signing engine used:

Signing Engine	Signer Setting
Generic RSA	common name, email
CryptoAPI	common name, email
Netscape	common name, email
Entrust	signer's login identity
CIC	signer's name as entered during signing ceremony
Clickwrap	*Accepted*

Signing Engine	Signer Setting
HMAC ClickWrap	the value of the answer indicated by the HMACSigner tag in the *signformat* option Note that if the HMACSigner tag includes more than one answer, they are combined in a comma delimited list. For example, “answer1, answer2”.

2. The *signer* option is automatically generated by the signature button when the user signs the form. It is added to both the signature button code and the signature code. No manual coding is required.

8.63 signformat

Records the type of encoding that a form viewing program must use to create the mimedata setting in a signature. Specifically, the parameters in signformat specify:

- The MIME type of the data from which the mimedata setting is created (see below for an explanation).
- The signature engine to use.
- Settings specific to the engine used.

About the mimedata setting:

To create the mimedata setting, a form viewer takes the signer's certificate and a plaintext representation of the form or portion of the form that the signature applies to, and encodes them according to the settings in signformat. For details, see the *mimedata* option.

Syntax

```
<signformat>MIME type;
   engine="signature engine";
   verifier;
   cval;
   parameters
</signformat>
```

MIMEtype	*string*	the MIME type of the signed data. May be: ■ **XFDL** — application/vnd.xfdl
signature engine	*string*	the type of signature. Valid types are: ■ Generic RSA (any standard RSA signature engine, including CryptoAPI and Netscape) ■ CryptoAPI ■ InkTools ■ Netscape ■ ClickWrap ■ HMAC-ClickWrap **Default**: Generic RSA

verifier	*string*	an optional flag that indicates which verifier should be used when verifying certificate chains during digital signature operations. Valid verifiers are: ■ Basic — Performs basic certificate verification. ■ DODJ12 — Performs strict certificate verification that complies with US Department of Defense requirements. **Default:** Basic
cval	*string*	an optional flag that indicates whether the current value of computed options is signed. This is useful if you want to sign the compute, but not the value calculated by the compute (for example, if you are signing the presentation layer of a form). If you want to sign the current values, do not use this flag. If you do not want to sign the current values, use: `cval="off"` **Default:** current values are signed.
parameters	*depends on engine*	additional parameters required by the signature engine (see below)

Available In

button, signature

Example

This example shows a button configured for a CryptoAPI signature.

```
<button sid="empSigButton">
   <type>signature</type>
   <value compute="signer"></value>
   <signer></signer>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <signformat>application/vnd.xfdl;
      csp="Microsoft Base Cryptographic Provider v1.0";
      csptype=rsa_full; hashalg=sha1
   </signformat>
```

```
        <signoptions>
            <ae>omit</ae>
            <ae>triggeritem</ae>
            <ae>coordinates</ae>
        </signoptions>
        <signitemrefs>
            <ae>omit</ae>
            <ae>PAGE1.mgrSigButton</ae>
            <ae>PAGE1.admSigButton</ae>
            <ae>PAGE1.empSignature</ae>
            <ae>PAGE1.mgrSignature</ae>
            <ae>PAGE1.admSignature</ae>
        </signitemrefs>

    <!-- The items listed above MUST have itemlocation
    options with absolute and extent as the last
    settings in order for the filter below to
    be sufficient in terms of security -->

        <signoptionrefs>
            <ae>keep</ae>
            <ae>PAGE1.mgrSigButton.itemlocation</ae>
            <ae>PAGE1.admSigButton.itemlocation</ae>
            <ae>PAGE1.empSignature.itemlocation</ae>
            <ae>PAGE1.mgrSignature.itemlocation</ae>
            <ae>PAGE1.admSignature.itemlocation</ae>
        </signoptionrefs>
        <signature>empSignature</signature>
    </button>
```

This example shows a button configured for a InkTools signature. The data item “SIGIMAGE” holds the image that is captured from the signature. The data item “BLANKIMAGE”” (not shown) holds an alternate image to display if the signature is not valid.

```
    <button sid="SignVacFormButton">
        <type>signature</type>
        <value>Sign Here</value>
        <image compute="(signer != ’’ andsigner != &#xA;
            ’INVALID’) ? ’SIGIMAGE’ : ’BLANKIMAGE’">BLANKIMAGE</image>
        <signer></signer>
        <format>
            <ae>string</ae>
            <ae>mandatory</ae>
        </format>
        <signformat>application/vnd.xfdl;
            engine="PenOp"; SignatureType="sign";
            SignatureText="I hereby authorize the vacation time
            indicated on this form"; IdentityPrompt="Please enter your
            full name and title."
```

```
        </signformat>
        <signoptions>
            <ae>omit</ae>
            <ae>triggeritem</ae>
            <ae>coordinates</ae>
        </signoptions>
        <signitemrefs>
            <ae>omit</ae>
            <ae>PAGE1.SignVacFormButton</ae>
            <ae>PAGE1.signaturedata</ae>
            <ae>PAGE1.mgrSigButton</ae>
            <ae>PAGE1.admSigButton</ae>
            <ae>PAGE1.empSignature</ae>
            <ae>PAGE1.mgrSignature</ae>
            <ae>PAGE1.admSignature</ae>
        </signitemrefs>

    <!-- The items listed above MUST have itemlocation
    options with absolute and extent as the last
    settings in order for the filter below to
    be sufficient in terms of security -->

        <signoptionrefs>
            <ae>keep</ae>
            <ae>PAGE1.mgrSigButton.itemlocation</ae>
            <ae>PAGE1.admSigButton.itemlocation</ae>
            <ae>PAGE1.empSignature.itemlocation</ae>
            <ae>PAGE1.mgrSignature.itemlocation</ae>
            <ae>PAGE1.admSignature.itemlocation</ae>
        </signoptionrefs>
        <signature>empSignature</signature>
    </button>

    <data sid="SIGIMAGE">
        <datagroup>images</datagroup>
        <mimetype>image/bmp</mimetype>
        <mimedata encoding="base64-gzip">data removed</mimedata>
    </data>
```

This example shows a button configured for a ClickWrap signature.

```
    <button sid="empSigButton">
        <type>signature</type>
        <value compute="signer"></value>
        <signer></signer>
        <format>
            <ae>string</ae>
            <ae>mandatory</ae>
        </format>
```

```
    <signformat>application/vnd.xfdl;
        engine="ClickWrap"; hashalg="md5";
        titleText="Document Acceptance";
        echoPrompt="Type the following:";
        echoText="I agree";question1Text="Name:";
        question2Text="Employee ID #:"
    </signformat>
    <signoptions>
        <ae>omit</ae>
        <ae>triggeritem</ae>
        <ae>coordinates</ae>
    </signoptions>
    <signitemrefs>
        <ae>omit</ae>
        <ae>PAGE1.mgrSigButton</ae>
        <ae>PAGE1.admSigButton</ae>
        <ae>PAGE1.empSignature</ae>
        <ae>PAGE1.mgrSignature</ae>
        <ae>PAGE1.admSignature</ae>
    </signitemrefs>

<!-- The items listed above MUST have itemlocation
    options with absolute and extent as the last
    settings in order for the filter below to
    be sufficient in terms of security -->

    <signoptionrefs>
        <ae>keep</ae>
        <ae>PAGE1.mgrSigButton.itemlocation</ae>
        <ae>PAGE1.admSigButton.itemlocation</ae>
        <ae>PAGE1.empSignature.itemlocation</ae>
        <ae>PAGE1.mgrSignature.itemlocation</ae>
        <ae>PAGE1.admSignature.itemlocation</ae>
    </signoptionrefs>
    <signature>empSignature</signature>
</button>
```

Usage Notes

1. An XFDL Viewer automatically copies the *signformat* option from a signature button to its associated signature item.
2. Signformat is an optional setting for a button item, but is mandatory for a signature item.
3. The list below describes the settings for additional parameters for some common signature engines. Note that ClickWrap and HMAC-ClickWrap signatures share the same parameters.

Additional Parameters for Common Signature Engines

Engine	Parameter	Valid Settings	Description
CryptoAPI	csp	determined by form viewing program	Cryptographic Service Provider (ie Microsoft Base Cryptographic Provider v1.0)
	csptype	rsa_full	full RSA implementation (this is the default)
		rsa_sig	for a CSP that supplies only RSA signature algorithms
		dss	for a CSP that supplies algorithms compliant with the Digital Signature Standard
		dss_dh	for a CSP that supplies DSS compliant algorithms and Diffie-Hellman encryption
		fortezza	for a CSP that supplies Fortezza algorithms
	hashalg	sha1	hash algorithm with 160-bit message digest. This is the default setting.
		md5	hash algorithm with 128-bit message digest
InkTools	SignatureType	Sign	user will sign form, signature will not be validated against CIC database. This is the default setting.
		SignAndVerify	user will sign form, signature will be verified against CIC database

Engine	Parameter	Valid Settings	Description
	IdentityPrompt	string	text to display prompting the user to enter his or her name, e-mail address or any other required information. Used only if SignatureType is Sign. Default: “Enter your name”.
	SignatureText	string	text to display that describes what the user is signing. Default: “As per form contents”.
Netscape	hashalg	sha1	hash algorithm with 160-bit message digest. This is the default setting.
		md5	hash algorithm with 128-bit message digest
ClickWrap	hashalg	sha1	hash algorithm with 160-bit message digest. This is the default setting.
		md5	hash algorithm with 128-bit message digest
	titleText	string	text to display for the main title of the signature dialog box
	mainPrompt	string	text to display for the main prompt
	mainText	string	text to display for the main text
	question1Text	string	label for the first question
	answer1Text	string	default answer to the first question; user can overwrite
	question2Text	string	label for the second question
	answer2Text	string	default answer to the second question; user can overwrite

Engine	Parameter	Valid Settings	Description
	question3Text	string	label for the third question
	answer3Text	string	default answer to the third question; user can overwrite
	question4Text	string	label for the forth question
	answer4Text	string	default answer to the forth question; user can overwrite
	question5Text	string	label for the fifth question
	answer5Text	string	default answer to the fifth question; user can overwrite
	echoPrompt	string	text to display for the echo prompt
	echoText	string	text to display for the signer to echo
	buttonPrompt	string	text to display above the accept and reject buttons
	acceptText	string	text to display on the accept button
	rejectText	string	text to display on the reject button
	HMACSigner	string	a comma delimited list of the the answers that store the signer's identity. This is written as $answer_n$. For example, $answer_1$, $answer_2$, and so on. Note that this parameter applies only to HMAC-ClickWrap signatures

Engine	Parameter	Valid Settings	Description
	HMACSecret	string	a comma delimited list of the the answers that store the shared secret. This is written as *answer*$_n$. For example, answer$_1$, answer$_2$, and so on. Note that this parameter applies only to HMAC-ClickWrap signatures
	readonly	string	a comma delimited list of the the answers that should be read-only. This is written as *answer*$_n$. For example, answer$_1$, answer$_2$, and so on. Note that this parameter applies only to HMAC-ClickWrap signatures
Entrust	hashalg	sha1	hash algorithm with 160-bit message digest. This is the default setting.
		md5	hash algorithm with 128-bit message digest

Note: *Instead of using one of the above settings for csptype, the numeric value that is defined for it in the cryptographic API may be used. For example,* csptype=dss *and* csptype=3 *produce the same result.*

8.64 signgroups

Specifies which groups are filtered for a particular signature. Filtering a group means keeping or omitting all the *cell* items that are in the specified group.

For example, if a *signgroups* option specifies that the “colorcells” group should be kept, then all cells within the “colorcells” group will be signed.

Syntax

```
<signgroups>
   <ae>group filter</ae>
   <ae>group reference1</ae>
   ...
   <ae>group referencen</ae>
</signgroups>
```

Note:

- There may be any number of *group reference* entries.

group filter	*keep*	include groups of cells in the *group reference* list with the signature; omit those not in the list
	omit	omit groups of cells in the *group reference* list from the signature; include those not in the list
group reference	*string*	identifies a group whose *cell* items will be filtered

Available In

button, signature

Example

This example shows a signgroups setting that omits the group of cells named “monthlyPayCells”.

```
<signgroups>
   <ae>omit</ae>
   <ae>monthlyPayCells</ae>
</signgroups>
```

Usage Notes

1. Default: **omit nothing (keep all cell items), unless the containing page is omitted**
2. It is possible to have several *list* or *popup* items with the same group reference, as these are populated with cells that have the same group reference as the item which contains them. Therefore, when setting up *signgroups* options, caution must be exercised in making group references to *list* or *popup* items which might be populated by the same group of cells.
3. Other filters may take precedence over the *signgroups* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.65 signitemrefs

Specifies individual items that are filtered for a particular signature. Filtering an item reference means keeping or omitting specific items, rather than all items of a particular type (see *signitems*).

Syntax

```
<signitemrefs>
   <ae>item filter</ae>
   <ae>item reference1</ae>
   ...
   <ae>item referencen</ae>
</signitemrefs>
```

Note:

- There may be any number of *item reference* entries.

item filter	*keep*	include items in the *item reference* list with the signature; omit those not in the list
	omit	omit items in the *item reference* list from the signature; include those not in the list
item reference	*string*	specifies the item to be filtered

Available In

button, signature

Example

This sample sets the *signitemrefs* option to omit two fields from the signature:

```
<signitemrefs>
   <ae>omit</ae>
   <ae>field1</ae>
   <ae>page1.field2</ae>
</signitemrefs>
```

Usage Notes

1. Default: **omit nothing (keep all items), unless the containing page is omitted**

2. Since all items have a name and type, signitemrefs filters are always applicable.
3. When not signing the entire form, it is *strongly recommended* that an **omit** signitemrefs filter be used to exclude unwanted items, and that a **keep** signoptionrefs filter should be used to cover the layout of omitted items. See the usage notes for the *signoptionrefs* option for details.
4. If this filter is used with the keep setting, then it is easy to add or delete items that would obscure or unobscure signed items without breaking a signature. The keep setting is useful as an optimization when you want to create a co-signature (i.e., a signature which signs another signature that uses omission logic).
5. Other filters may take precedence over the *signitemrefs* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.66 signitems

Specifies which types of items filtered for a particular signature. Filtering an item means keeping or omitting all items of a particular type, rather than specific items (see *signitemrefs*).

Syntax

```
<signitems>
   <ae>item filter</ae>
   <ae>item type1</ae>
   ...
   <ae>item typen</ae>
</signitems>
```

Note:

- There may be any number of *item type* entries.

item filter	*keep*	include types of items in the *item type* list with the signature; omit those not in the list
	omit	omit types of items in the *item type* list from the signature; include those not in the list
item type	*string*	specifies the type (element tag name) of items to be filtered

Available In

button, signature

Example

This sample sets the signitems option to keep the following types of items with the signature: boxes, buttons, and fields.

```
<signitems>
   <ae>keep</ae>
   <ae>box</ae>
   <ae>button</ae>
   <ae>field</ae>
</signitems>
```

Usage Notes

1. Default: **omit nothing (keep all items), unless the containing page is omitted**
2. The only recommended use of this filter is to omit *data* items in support of the usage of the signdatagroups filter.
3. Element tag names can be specified with or without a namespace prefix. The default namespace URI is the XFDL namespace URI if no namespace prefix is given. If a namespace prefix is given, then the namespace URI to which the prefix is bound is used in determining whether or not each element matches the given filter.

   For example, the filter component *<ae>box</ae>* could equivalently be written *<ae>xfdl:box</ae>* if the prefix *xfdl* is mapped to the XFDL namespace URI.
4. Other filters may take precedence over the *signitems* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.67 signnamespaces

Specifies which namespaces are filtered for a particular signature. Filtering a namespace means keeping or omitting all of the form elements and attributes that are in the specified namespace.

For example, if a *signnamespaces* option specifies that the http://www.PureEdge.com/XFDL/Custom namespace should be kept, then all elements in that namespace are signed.

Syntax

```
<signnamespaces>
   <ae>namespace filter</ae>
   <ae>namespace URI1</ae>
   ...
   <ae>namespace URIn</ae>
</signnamespaces>
```

Note:

- There may be any number of *namespace URI* entries.

namespace filter	*keep*	include all form elements in the *namespace URI* list with the signature; omit those not in the list
	omit	omit all form element that are in the namespaces in the *namespace URI* list from the signature; include those not in the list
namespace URI	*string*	identifies a namespace whose elements will be filtered

Available In

button, signature

Example

This example shows a *signnamespaces* setting that omits the http://www.PureEdge.com/XFDL/Custom namespace.

```
<signnamespaces>
   <ae>omit</ae>
   <ae>http://www.PureEdge.com/XFDL/Custom</ae>
</signnamespaces>
```

Usage Notes

1. Default: **omit nothing (keep all namespaces)**
2. Other filters may take precedence over the *signnamespaces* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.68 signoptionrefs

Specifies individual options that are filtered for a particular signature. Filtering option references means keeping or omitting specific options, rather than all options of a particular type (see *signoptions*).

Syntax

```
<signoptionrefs>
   <ae>option filter</ae>
   <ae>option reference1</ae>
   ...
   <ae>option referencen</ae>
</signoptionrefs>
```

Note:

- There may be any number of *option reference* entries.

option filter	*keep*	include options in the *option reference* list with the signature; omit those not in the list
	omit	omit options in the *option reference* list from the signature; include those not in the list
option reference	*string*	specifies the option to be filtered

Available In

button, signature

Example

This example specifies a signoptionrefs setting that keeps a particular field with the signature.

```
<signoptionrefs>
   <ae>keep</ae>
   <ae>page1.field1.value</ae>
</signoptionrefs>
```

Note: *The page name may be dropped if the option in question is on the same page, but the item name must not be dropped.*

Usage Notes

1. Default: **keep all options (omit nothing), unless the containing item is omitted**
2. Note that, unlike signoptions, the signoptionrefs filter can cause an item to be included even if the item filters would normally omit the item. This is necessary in order to ensure that the hashed text of a signature is in valid XFDL format.
3. It is *strongly recommended* that signoptionrefs be used to **keep** the itemlocation of items that have been omitted, and that all omitted items have an itemlocation with absolute and extent settings as the last two settings.
4. Element tag names can be specified with or without a namespace prefix. The default namespace URI is the XFDL namespace URI if no namespace prefix is given. If a namespace prefix is given, then the namespace URI to which the prefix is bound is used in determining whether or not each element matches the given filter.

   For exampe, the filter component *<ae>page1.field1.value</ae>* could equivalently be written *<ae>page1.field1.xfdl:value</ae>* if the prefix *xfdl* is mapped to the XFDL namespace URI.
5. Other filters may take precedence over the *signoptionrefs* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.69 signoptions

Specifies which types of options are filtered for a particular signature. Filtering options means keeping or omitting all options of a particular type, rather than specific options (see *signoptionrefs*).

Syntax

```
<signoptions>
   <ae>option filter</ae>
   <ae>option type1</ae>
   ...
   <ae>option typen</ae>
</signoptions>
```

Note:

- There may be any number of *option type* entries.

option filter	*keep*	include types of options in the *option type* list with the signature; omit those not in the list
	omit	omit types of options in the *option type* list from the signature; include those not in the list
option type	*string*	specifies the type (element tag name) of options to be filtered

Available In

button, signature

Example

This example shows a signoptions setting that omits two types of options from the signature.

```
<signoptions>
   <ae>omit</ae>
   <ae>url</ae>
   <ae>printsettings</ae>
</signoptions>
```

Usage Notes

1. Default: **keep all options (omit nothing), unless the containing item is omitted**
2. One signoptions setting must always be specified in the following way:

   ```
   <signoptions>
      <ae>omit</ae>
      <ae>triggeritem</ae>
      <ae>coordinates</ae>
   </signoptions>
   ```

   - This setting ensures that the signature will not be broken due to an alteration to the form.
   - If the this option must be used as a keep filter, then consider omitting the *global.global.triggeritem* and the coordinates of image buttons using a signoptionrefs filter.
3. Element tag names can be specified with or without a namespace prefix. The default namespace URI is the XFDL namespace URI if no namespace prefix is given. If a namespace prefix is given, then the namespace URI to which the prefix is bound is used in determining whether or not each element matches the given filter component.

   For example, the filter component *<ae>triggeritem</ae>* could be equivalently written as *<ae>xfdl:triggeritem</ae>* if the prefix *xfdl* is mapped to the XFDL namespace URI.
4. Other filters may take precedence over the *signoptions* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.70 signpagerefs

Specifies individual pages that are filtered for a particular signature. Filtering pages means keeping or omitting a page and all of its contents.

Syntax

```
<signpagerefs>
   <ae>page filter</ae>
   <ae>page reference1</ae>
   ...
   <ae>page referencen</ae>
</signpagerefs>
```

Note:

- There may be any number of *page reference* entries.

page filter	*keep*	include pages in the *page reference* list with the signature; omit those not in the list
	omit	omit pages in the *page reference* list from the signature; include those not in the list
page reference	*string*	specifies the page to be filtered

Available In

button, signature

Example

This sample sets the signpagerefs option to omit two pages from the signature.

```
<signpagerefs>
   <ae>omit</ae>
   <ae>page1</ae>
   <ae>page2</ae>
</signpagerefs>
```

Usage Notes

1. Default: **keep all pages**
2. Other filters may take precedence over the *signpagerefs* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.71 size

Specifies an item's size. It does not include external labels, borders, or scroll bars. These are part of the bounding box size which is calculated automatically. The unit of measurement is characters.

Examples of item size are the input area in a *field* item or the height and width of the label in *label* and *button* items.

Syntax

```
<size>
   <ae>width</ae>
   <ae>height</ae>
</size>
```

width	*unsigned byte*	the horizontal dimension of the item
height	*unsigned byte*	the vertical dimension of the item

Available In

box, button, check, combobox, field, label, line, list, popup, radio, spacer

Example

This sample sets the item's size to 80 characters wide by five characters high.

```
<size>
   <ae>80</ae>
   <ae>5</ae>
</size>
```

Usage Notes

1. Size and Font:
   - The width might not always accommodate the number of characters specified. The calculation to determine actual width is:
     - *width* * (average character width for the item's font)
   - The width will only exactly match the number of characters the item can display horizontally when the font is mono-spaced (like Courier).
2. If either the height or the width is invalid, the default item size will be used. A dimension of zero (0) is invalid for all items except the *line* item.

3. The item and bounding box sizes can be changed by using *itemlocation* with an **expansion** or **extent** modifier. This will override the *size* option.

8.72 texttype

Specifies whether a field uses plain text or rich text.

Syntax

```
<texttype>type</texttype>
```

type	*string*	the type of text to use. Valid options are: ■ text/plain ■ text/rtf

Available In

field

Example

This sample sets a field to use rich text.

```
<field sid="richTextField">
   <texttype>text/rtf</texttype>
   <value>Hello</value>
   <rtf>rich text version of Hello</rtf>
</field>
```

Usage Notes

1. Default: **text/plain**
2. If using rich text:
   - The rich text is stored in the *rtf* option.
   - A plain text version of the rich text is stored in the *value* option (refer to the *rtf* option for more information).

8.73 thickness

Specifies the thickness of a *line* item. The unit of measurement is pixels.

Syntax

```
<thickness>thickness</thickness>
```

thickness	*unsigned byte*	the thickness of the line

Available In

line

Example

This sample defines a horizontal line 40 characters long and five pixels thick:

```
<size>
   <ae>40</ae>
   <ae>0</ae>
</size>
<thickness>5</thickness>
```

Usage Notes

1. Default: **one pixel**
2. Use size to define the dimension of a line in one direction (height or width) and thickness to define the dimension in the other direction. The dimension thickness defines must be set to **zero** in size.
3. The line's thickness may be changed by using itemlocation with an **expansion** modifier for the dimension that thickness describes.

8.74 transmitdatagroups

Specifies which datagroups are filtered when the form is transmitted. Filtering a datagroup means keeping or omitting all the *data* items that are in the specified datagroup.

For example, if a *transmitdatagroups* option specifies that the “attachments” datagroup should be kept, then all data items within the “attachments” group will be transmitted.

This filter applies to all *data* items present when transmitted, including those added as enclosures.

Syntax

```
<transmitdatagroups>
   <ae>datagroup filter</ae>
   <ae>datagroup reference1</ae>
   ...
   <ae>datagroup referencen</ae>
</transmitdatagroups>
```

Note:

- There may be any number of *datagroup reference* entries.

datagroup filter	*keep*	include datagroups in the *datagroup reference* list with the transmission; omit those not in the list
	omit	omit datagroups in the *datagroup reference* list from the transmission; keep those not in the list
datagroup reference	*string*	identifies a datagroup whose *data* items will be filtered

Available In

action, button, cell

Examples

This sample specifies that only items of type data with a datagroup setting of enclosures should be transmitted:

```
<transmitdatagroups>
   <ae>keep</ae>
   <ae>enclosures</ae>
```

```
</transmitdatagroups>
```

This sample specifies that all items of type data except those with a datagroup setting of **others** should be kept in the transmission:

```
<transmitdatagroups>
   <ae>omit</ae>
   <ae>others</ae>
</transmitdatagroups>
```

Usage Notes

1. Default: **omit nothing (keep all data items), unless the containing page is omitted**
2. Since enclosed files can belong to several datagroups, and datagroups can contain several enclosed files, care must be exercised when setting up transmitdatagroups options to ensure that only the desired datagroups are filtered.
3. Other filters may take precedence over the *transmitdatagroups* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.75 transmitformat

Specifies the format of the form data submitted to a processing application. An XFDL form can submit data in XFDL format or in HTML format. Furthermore, the XFDL format may be compressed using ASCII compression.

XFDL format submits the entire form definition, including user input.

HTML format submits just an assignment statement for each item equating the item reference with the item's value. The only items included are modifiable items, custom items, and items with a *transmit* option setting of **all**.

Note: *Form and page globals are sent only if the format is XFDL.*

Syntax

<transmitformat>*MIME type*</transmitformat>

MIME type	*application/vnd.xfdl*	use XFDL format
	application/vnd.xfdl; content-encoding="base64-gzip"	use compressed XFDL format
	application/x-www-form-urlencoded	use HTML form format

Available In

action, button, cell, page global, form global

Examples

XFDL format

This example shows a button which, when clicked, will submit the form in XFDL format.

```
<button sid="send_button">
   <type>done</type>
   <url>
      <ae>mailto:rrunner@acme.com</ae>
   </url>
   <transmitformat>application/vnd.xfdl</transmitformat>
</button>
```

When a user clicks the button, the entire form definition will be submitted, unless other transmit options specify a partial submission.

HTML form format

This sample shows an automatic action that submits form data in HTML form format.

```
<action sid="status_action">
   <type>submit</type>
   <url>
      <ae>http://www.host.domain/cgi-bin/recvStatus</ae>
   </url>
   <transmitformat>
      application/x-www-form-urlencoded
   </transmitformat>
   <delay>
      <ae>repeat</ae>
      <ae>180</ae>
   </delay>
</action>
```

Every 180 seconds, the form definition will be converted to HTML form format.

HTML Format by Item Type

The general syntax of a submitted HTML form is:

```
item reference=value&item reference=value&...
```

Note:

- The ampersand separates form items.

The syntax of an HTML form entry by item type:

Item Type	HTML Format
field	item tag=value option setting
list	item tag=value option setting of selected cell Note that the item reference identifies the list.

Item Type	HTML Format
popup	item tag=value option setting of selected cell Note that the item reference identifies the popup.
combobox	item tag=value option setting
check	item tag=value option setting
radio	group option setting=item tag of selected radio
custom	item tag=value option setting
all other items	item tag=value option setting

Substitutions and Omissions:

- Spaces in the value are replaced by the plus sign (+). For example, ‘Two words’ becomes ‘Two+words’.
- The membership operator in item and group references is replaced by a minus sign.
- *page_one.age_group* becomes *page_one-age_group*.
- Page tags are removed from item and group references in single page forms.
- Check boxes and radio buttons with a *value* option setting of **off** are omitted.
- Entries resulting in an empty string on the right hand side of the assignment statement are omitted. This occurs when the referenced option setting is empty or the option definition is missing.

HTML Considerations

The functionality of XFDL forms differs somewhat from HTML forms. Those differences are:

- **Enclosures** — HTML does not support enclosures. To submit enclosed form data, use XFDL format.
- **Item tags** — XFDL allows a smaller set of characters in item tags than HTML does. XFDL item tags support the following characters: a-z, A-Z, 0-9, and the underscore (_).
- **Check boxes** — XFDL check boxes vary slightly from HTML check boxes. XFDL check boxes are independent items; HTML check boxes are grouped together using the same format as *radio* items. When an XFDL form is submitted in HTML format, the submission will contain an entry for each check box.

Usage Notes

1. Default: The default is the format that the form was in before it was parsed. For example, a form written in XFDL will be transmitted in XFDL unless otherwise specified by this option.

2. This option can be included as a form global option and in the definitions of items that trigger form submissions. These items have a *type* option setting of **submit** or **done**.

8.76 transmitgroups

Specifies which groups are filtered for a particular trasmission. Filtering a group means keeping or omitting all the *cell* items that are in the specified group.

For example, if a *transmitgroups* option specifies that the “colorcell” group should be kept, then all cells within the “colorcell” group will be transmitted.

Syntax

```
<transmitgroups>
   <ae>transmitflag</ae>
   <ae>group reference1</ae>
   ...
   <ae>group referencen</ae>
</transmitgroups>
```

Note:

- There may be any number of *group reference* entries.

transmit flag	*keep*	include groups of cells in the *group reference* list with the transmission; omit those not in the list
	omit	omit groups of cells in the *group reference* list from the transmission; include those not in the list
group reference	*string*	identifies a group whose *cell* items will be filtered

Available In

action, button, cell

Examples

This sample specifies that only the items in the “countryCells” and “departmentCells” groups should be kept in the transmission.

```
<transmitgroups>
   <ae>keep</ae>
   <ae>countryCells</ae>
   <ae>departmentCells</ae>
</transmitgroups>
```

This sample specifies that all groups of cells should be kept in the transmission except for “firstNameCells” group.

```
<transmitgroups>
   <ae>omit</ae>
   <ae>firstNameCells>/ae>
</transmitgroups>
```

Usage Notes

1. Default: **omit nothing (keep all cell items), unless the containing page is omitted**
2. It is possible to have several *list* or *popup* items with the same group reference, as these are populated with cells that have the same group reference as the item which contains them. Therefore, when setting up a *transmitgroups* option, caution must be exercised in making group references to *list* or *popup* items which might be populated by the same group of cells.
3. Other filters may take precedence over the *transmitgroups* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.77 transmititemrefs

Specifies individual items that are filtered for a particular transmission. Filtering item references means keeping or omitting individual items, rather than all items of a particular type (see *transmititems*).

Syntax

```
<transmititemrefs>
   <ae>transmit flag</ae>
   <ae>item reference1</ae>
   ...
   <ae>item referencen</ae>
</transmititemrefs>
```

Note:

- There may be any number of item reference entries.

transmit flag	*keep*	include items in the *item reference* list with the transmission; omit those not in the list
	omit	omit items in the *item reference* list from the transmission; include those not in the list
item reference	*string*	identifies the item to be filtered

Available In

action, button, cell

Examples

This sample specifies that only the item on page1 called “MgrSignButton” should be transmitted, and that all other items should be omitted.

```
<transmititemrefs>
   <ae>keep</ae>
   <ae>page1.MgrSignButton</ae>
</transmititemrefs>
```

This sample shows how you would use transmititemrefs in conjunction with transmititems: although all items that are buttons are omitted, the button on page1 called “MgrSignButton” will be kept.

```
<transmititems>
```

```
    <ae>omit</ae>
    <ae>button</ae>
</transmititems>
<transmititemrefs>
    <ae>keep</ae>
    <ae>page1.MgrSignButton</ae>
</transmititemrefs>
```

Usage Notes

1. Default: **omit nothing (keep all items), unles the containing page is omitted**
2. Other filters may take precedence over the *transmititemrefs* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.78 transmititems

Specifies types of items that are filtered for a particular transmission. Filtering items means keeping or omitting all items of a particular type, rather than individual items (see *transmititemrefs*).

Syntax

```
<transmititems>
   <ae>transmit flag</ae>
   <ae>item type1</ae>
   ...
   <ae>item typen</ae>
</transmititems>
```

Note:

- There may be any number of *item type* entries.

transmit flag	*keep*	include types of items in the *item type* list with the transmission; omit those not in the list
	omit	omit types of items in the *item type* list from the transmission; include those not in the list
item type	*string*	identifies the type (element tag name) of items to be filtered

Available In

action, button, cell

Example

This sample specifies that *box*, *help*, *label*, *spacer*, and *toolbar* items should be omitted from the form data submitted to the form processing application.

```
<transmititems>
   <ae>omit</ae>
   <ae>box</ae>
   <ae>help</ae>
   <ae>spacer</ae>
```

```
    <ae>toolbar</ae>
</transmititems>
```

Usage Notes

1. Default: **omit nothing (keep all items), unless the containing page is omitted**
2. Element tag names can be specified with or without a namespace prefix. The default namespace URI is the XFDL namespace URI if no namespace prefix is given. If a namespace prefix is given, then the namespace URI to which the prefix is bound is used in determining whether or not each element matches the given filter.

   For example, the filter component *<ae>box</ae>* could equivalently be written *<ae>xfdl:box</ae>* if the prefix *xfdl* is mapped to the XFDL namespace URI.
3. Other filters may take precedence over the *transmititems* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.79 transmitnamespaces

Specifies which namespaces are filtered for a particular transmission. Filtering a namespace means keeping or omitting all of the form elements and attributes that are in the specified namespace.

For example, if a *transmitnamespaces* option specifies that the http://www.PureEdge.com/XFDL/Custom namespace should be kept, then all elements in that namespace are transmitted.

Syntax

```
<transmitnamespaces>
   <ae>namespace filter</ae>
   <ae>namespace URI1</ae>
   ...
   <ae>namespace URIn</ae>
</transmitnamespace>
```

Note:

- There may be any number of *namespace URI* entries.

namespace filter	*keep*	include all form elements in the *namespace URI* list with the transmission; omit those not in the list
	omit	omit all form element that are in the namespaces in the *namespace URI* list from the transmission; include those not in the list
namespace URI	*string*	identifies a namespace whose elements will be filtered

Available In

action, button, cell

Example

This example shows a *transmitnamespaces* setting that omits the http://www.PureEdge.com/XFDL/Custom namespace.

```
<transmitnamespaces>
   <ae>omit</ae>
   <ae>http://www.PureEdge.com/XFDL/Custom</ae>
</transmitnamespaces>
```

Usage Notes

1. Default: **omit nothing (keep all namespaces)**
2. Other filters may take precedence over the *transmitnamespaces* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.80 transmitoptionrefs

Specifies individual options that are filtered for a particular transmission. Filtering option references means keeping or omitting individual options, rather than all options of a particular type (see *transmitoptions*).

Syntax

```
<transmitoptionrefs>
   <ae>transmit flag</ae>
   <ae>option reference1</ae>
   ...
   <ae>option referencen</ae>
</transmitoptionrefs>
```

Note:

- There may be any number of *option reference* entries.

transmit flag	*keep*	include options in the *option reference* list with the transmission; omit those not in the list
	omit	omit options in the *option reference* list from the transmission; include those not in the list
option reference	*string*	identifies the option to be filtered

Available In

action, button, cell

Examples

This sample shows how you would use transmitoptionrefs in conjunction with transmitoptions: although all options that are *values* are omitted, the value in the “NameField” on “page1” will be kept.

```
<transmitoptions>
   <ae>omit</ae>
   <ae>value</ae>
</transmitoptions>
<transmitoptionrefs>
   <ae>keep</ae>
   <ae>page1.NameField.value</ae>
```

```
</transmitoptionrefs>
```

This sample shows how you would use transmitoptionrefs in conjunction with transmititemrefs: although the item called “MgrSignButton” on “page1” is omitted, its *signer* option is kept

```
<transmititemrefs>
   <ae>omit</ae>
   <ae>MgrSignButton</ae>
</transmititemrefs>
<transmitoptionrefs>
   <ae>keep</ae>
   <ae>page1.MgrSignButton.signature</ae>
</transmitoptionrefs>
```

Usage Notes

1. Default: **keep all options (omit nothing), unless the containing item is omitted**
2. Element tag names can be specified with or without a namespace prefix. The default namespace URI is the XFDL namespace URI if no namespace prefix is given. If a namespace prefix is given, then the namespace URI to which the prefix is bound is used in determining whether or not each element matches the given filter.

   For exampe, the filter component *<ae>page1.field1.value</ae>* could equivalently be written *<ae>page1.field1.xfdl:value</ae>* if the prefix *xfdl* is mapped to the XFDL namespace URI.
3. Other filters may take precedence over the *transmitoptionrefs* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.81 transmitoptions

Specifies types of options that are filtered for a particular transmission. Filtering options means keeping or omitting all options of a particular type, rather than individual items (see *transmititemrefs*).

Syntax

```
<transmitoptions>
   <ae>transmit flag</ae>
   <ae>option type1</ae>
   ...
   <ae>option typen</ae>
</transmitoptions>
```

Note:

- There may be any number of *option type* entries.

transmit flag	*keep*	include types of options in the *option type* list with the transmission; omit those not in the list
	omit	omit types of options in the *option type* list from the transmission; include those not in the list
option type	*string*	specifies the type (element tag name) of options to be filtered

Available In

action, button, cell

Example

This sample specifies that only the *active*, *mimedata*, and *value* options should be included in the form data submitted to the form processing application.

```
<transmitoptions>
   <ae>keep</ae>
   <ae>active</ae>
   <ae>mimedata</ae>
   <ae>value</ae>
</transmitoptions>
```

Usage Notes

1. Default: **keep all options (omit nothing), unless the containing item is omitted**
2. Element tag names can be specified with or without a namespace prefix. The default namespace URI is the XFDL namespace URI if no namespace prefix is given. If a namespace prefix is given, then the namespace URI to which the prefix is bound is used in determining whether or not each element matches the given filter component.

   For example, the filter component *<ae>triggeritem</ae>* could be equivalently written as *<ae>xfdl:triggeritem</ae>* if the prefix *xfdl* is mapped to the XFDL namespace URI.
3. Other filters may take precedence over the *transmitoptions* option. Refer to "Appendix C: Order of Precedence of Filters" on page 381 for more information on the order of precedence of filters.

8.82 transmitpagerefs

Specifies individual pages that are filtered for a particular signature. Filtering pages means keeping or omitting a page and all of its contents.

Syntax

```
<transmitpagerefs>
   <ae>transmit flag</ae>
   <ae>page reference1</ae>
   ...
   <ae>page referencen</ae>
</transmitpagerefs>
```

Note:

- There may be any number of *page reference* entries.

transmit flag	*keep*	include pages in the *page reference* list with the transmission; omit those not in the list
	omit	omit pages in the page reference list from the transmission; include those not in the list
page reference	*string*	specifies the page to be filtered

Available In

action, button, cell

Examples

This sample specifies that only page1 should be transmitted, and that all other pages should be omitted:

```
<transmitpagerefs>
   <ae>keep</ae>
   <ae>page1</ae>
</transmitpagerefs>
```

Usage Notes

1. Default: **keep all pages**

2. Other filters may take precedence over the *transmitpagerefs* option. Refer to “Appendix C: Order of Precedence of Filters” on page 381 for more information on the order of precedence of filters.

8.83 triggeritem

Identifies the item that triggered a form submission. Items triggering form submissions have a *type* option setting of **refresh**, **submit**, or **done**.

When a user selects an item that triggers a form submission, the *triggeritem* option is added to the form globals and assigned the item reference of the selected item.

Syntax

```
<triggeritem>item reference</triggeritem>
```

item reference	*string*	identifies the trigger item

Available In

form global

Example

This sample indicates that the item triggering the request is on the page called “Page_one” and has is called “submit_button”.

```
<triggeritem>Page_one.submit_button</triggeritem>
```

Usage Notes

1. Actions of type **submit** or **done** set the *triggeritem* to the SID of the triggering item. Actions of type **refresh** first clear the *triggeritem* by setting it to empty (""), then set the *triggeritem* to the SID of the triggering item.

8.84 type

Associates an action type with an item that can trigger a task: *action*, *button*, or *cell*.

Syntax

```
<type>action type</type>
```

action type *(see below)* the task to perform

Action Types

The action type can be any of the following:

Action Type	Description of Action Type	Use with the Following Items
link	Perform all requests specified by the url options in the current item. See section “8.85 url” for more details.	action, button, cell
replace	Perform a link followed by a cancel.	action, button, cell
submit	Initiate the form processing applications identified in the url options of the current item.	action, button, cell
done	Perform a submit followed by a cancel.	action, button, cell
pagedone	Move to the page specified in the url option. This closes the current page and replaces it with the new page. All fields containing error checking on the current page must be correctly filled out before it can be closed.	action, button, cell
cancel	Close the form; if any changes were made to the form since the last save or submit, then the user is informed that the form has changed and is allowed to choose whether the cancellation will proceed. Note that the value options of many items, as well as the contents of data items, may change in response to an enclose or remove action.	action, button, cell

Action Type	Description of Action Type	Use with the Following Items
refresh	Sets the triggeritem to "" and then to the full reference (including scope ID) of the item that triggered the refresh.	action, button, cell
save	Save the form in a file specified by the user.	button, cell
select	With cell items: flag the cell as selected when a user chooses the cell. This means the item reference of the cell is copied to the value option of the parent list or popup. With button items containing images: store coordinates of the mouse click position in the image into the coordinates option	button, cell
enclose	Allows the user to place one or more files into one or more of the datagroups defined for the form. The files will be encoded using base64 encoding format.	button, cell
extract	Allows a user to copy the contents of an enclosed file into a file on the local disk.	button, cell
remove	Allows the user to remove an item from a datagroup; the underlying data item will only be deleted if it belongs to no other datagroups.	button, cell
display	Display an enclosed file. The web browser will choose the appropriate viewer according to the file's MIME type.	action, button cell
print	Print the form on a local printer.	action, button, cell
signature	Create a signature.	button

Available In

action, button, cell

Example

This sample specifies that "BUTTON1" saves the form to a local file.

```
<button sid="BUTTON1">
   <value>Save</value>
   <type>save</type>
```

```
</button>
```

Usage Notes

1. Default: **select**

8.85 url

Provides the url to a target, such as a file or application. Items containing this option must have a *type* option setting of **link**, **replace**, **submit**, **done**, or **pagedone**.

The object identified must be one of the following:

- **File** — Used with a *type* option of **link** or **replace**. The file identified is downloaded, and either displayed or saved. Examples of such files are images, word processing documents, and XFDL forms.
- **Application** — Used with a *type* option of **submit** or **done**. The application identified is initiated. A form processing application, such as a cgi or a servlet, is an example of such an application.
- **Item** — Used with a *type* option of **pagedone**. The item identified, on the page identified, receives focus. The item must be on another page.
- **Form or Page Globals** — Used with a *type* option setting of **pagedone**. The focus moves to the item defined to receive focus when the form or page appears. The form globals reference is global.global. The page globals reference is <page tag>.global for another page.
- **Email Address** — Used with a *type* option of **submit**, **done**, **link**, or **replace**. With a **submit** or **done** type, the form is attached to an email message, and that message is sent to the address in the url. With a **link** or **replace** type, an email message is created and sent, but the form is not attached to the message. Depending on the settings you use, the user may be able to add additional information to the email.

Syntax

```
<url>
   <ae>the URL1</ae>
   ...
   <ae>the URLn</ae>
</url>
```

where the URL is one of:

- A URL with the format: scheme://host.domain[:port]/path/filename for files and applications. *Scheme* is restricted to http or https.
- A URL with a *mailto* format. See “URLs for Email” below for further information.
- *#item reference* for the next item in the form to receive focus.

Notes:

- The *URL* can occur 1 to n times.
- *item reference* can be a form or page global reference.

the URL	*string*	identifies the target

URLs for Email

URLs that provide an email address must follow this general format:

```
mailto:address?parameter=setting&parameter=setting...
```

The first parameter follows the question mark (?) symbol, while each additional parameter is added using the ampersand (&) symbol.

For example, a URL using all parameters would look like this:

```
mailto:setting?to=setting&cc=setting&bcc=setting&
subject=setting&body=setting&filename=setting
```

The following table lists the available parameters and their settings:

Parameter	Setting	Description
mailto: to= cc= bcc=	*string*	a complete email address, such as john@acme.com. To include additional addresses, use the appropriate parameter twice. For example, to add two cc addresses, use the **cc=** parameter twice as shown: `mailto:john@acme.com?cc=bob@acme.com&cc=fred@acme.com` Note that the first address, immediately after the **mailto:** parameter, is the first recipient. Additional recipients are specified using the **to=** parameter.
subject=	*string*	this is the subject line of the email. The text must conform to standard URL encoding rules, such as replacing spaces with the plus (+) symbol.
body=	*string*	this is the body of the email. The text must conform to standard URL encoding rules, such as replacing spaces with the plus (+) symbol.
filename=	*string*	this is the name you want to give to the file that is attached to the email message. If you do not set this parameter, a default file name will be assigned.

If you provide the **mailto:**, **cc=**, **bcc=**, **subject=**, and **body=** parameters, the email will be sent automatically and the user will not be able to modify the message. This is true even if the parameters are set to nothing. For example, the following URL would mail the message automatically:

```
mailto:tim@acme.com?cc=&bcc=&subject=Hello&body=Hello+Tim&body=
```

If you leave out any of those parameters, the user will see the email message before it is sent, and will be able to change the email.

Available In

action, button, cell

Example

This sample identifies a form processing application:

```
<url>
   <ae>http://www.host.domain/cgi-bin/recv_status</ae>
</url>
```

This sample identifies a page to display and an item on it to direct the focus to:

```
<url>
   <ae>#page_2.expense_field</ae>
</url>
```

This sample creates an email message that is sent automatically because it contains all of the necessary parameters. Note that the URL is enclosed in the CDATA construct because it contains ampersands (&).

```
<url>
   <ae>
      <![CDATA[mailto:john@acme.com?&subject=Hello&
      body=Hello.+How+are+you?]]>
   </ae>
</url>
```

This sample creates an email message that appears to the user before sending, allowing the user to change the parameters.

```
<url>
   <ae>mailto:john@acme.com</ae>
</url>
```

Usage Notes

1. Default: **none**
2. If you list multiple URLs, the URLs are processed in the order listed unless an action occurs that replaces or closes the form (for example, a *replace* or *done* action). Once the original form is closed by a *replace* or *done*, no further URLs are processed.

For example, if a button of type *submit* listed 2 URLs, the submission would process the first URL then the second URL. However, if the same button was of type *done*, the submission would process the first URL then close the original form, so the second URL would not be processed.

3. To create a URL that includes computed values, you must put the URL in a computed array element. For example:

```
<url>
   <ae compute="PAGE1.FIELD1.value"></ae>
</url>
```

4. To create a URL that includes user input as part of the URL string, you must place the CDATA construct within a computed array element. Ensure that you concatenate (+.) the elements of the string. Additionally, you must contain the email parameters (mailto, &subject, and so on) within quotation marks. For example:

```
<url>
   <ae compute="'mailto:' +. to_field.value +.&#xA;
      '&amp;subject=' +. subject_field.value +. '&amp;body=' &#xA;
      +.  body_field.value></ae>
</url>
```

5. If you have specified an HTML transmitformat in a form, the form sends its data as HTML when it communicates with a server. Information transmitted in HTML is URL-encoded. Therefore, for forms transmitted in HTML, you must replace all non alpha-numeric characters with a character triplet consisting of the % character followed by two hexadecimal digits. These hexadecimal digits are derived from the ASCII code for the original character. The hexadecimal digits are “0123456789ABCDEF”. For example:

Character	ASCII Code	URL-encoded triplet
<space>	32	%20
\r	13	%0D

Applications receiving form data must check the content type of the incoming data to see whether it is url-encoded.

8.86 value

Reflects the contents of an item. Visually, this can take several forms, depending on the item to which it applies. For example, the *value* option in *label* items contains the label text; the *value* option in *radio* items contains a status indicator; and the *value* option in *list* items contains the scope identifer (sid) of the most recently selected cell (if it was a select cell).

An item's contents will be stored in the form whenever a user saves the form or submits it for processing. This is true even for inactive items and items using the default *value* option setting (in this case, a *value* option containing the default setting is added to the item's definition).

Syntax

```
<value>setting</value>
```

setting	*string*	the item's contents

Available In

button, cell, check, combobox, field, help, label, list, popup, radio

Example

This sample identifies the text of a label item.

```
<value>My Form Title</value>
```

Usage Notes

1. Default: varies by item. Refer to the item in question for more information.
2. Rich text fields use both the *value* and the *rtf* option to store the text of the field. Refer to the *rtf* option for more information.
3. Multiple line values need to have carriage returns inserted in the code. For example:

   ```
   <value>This value spans
   two lines.</value>
   ```

4. To get the value of a cell that a user has selected from a list or popup, dereference it in the following manner:

   ```
   page_tag.list_tag.value->value
   ```

   For example:

```
page1.countryPopup.value->value
```

When a user selects a cell from a list, the scope identifer (sid) of the cell is stored as the value of the list. Hence the dereference syntax.

8.87 visible

Defines whether or not the item is visible on the screen and can be printed.

Syntax

```
<visible>status</visible>
```

status	*on*	item can be seen on the screen and printed
	off	item cannot be seen on the screen and will not print when the form is printed.

Available In

box, button, check, combobox, field, label, line, list, popup, radio

Example

This sample shows how an item can be set to be visible at the user's request.

```
<check sid="SHOW_INSTRUCTIONS">
   <value>off</value>
   <label>Do you want to see the instructions?</label>
</check>
<label sid="INSTRUCTION_LABEL">
   <visible compute="SHOW_INSTRUCTIONS.value=='on' ? &#xA;
      'on' : 'off'"></visible>
   <value>Please complete all portions of this form.</value>
</label>
```

Usage Notes

1. Default: **on**

8.88 <custom option>

Allows form designers to add application specific information to the form definition. This is useful when submitting forms to applications requiring non-XFDL information. An example of non-XFDL information might be an SQL query statement. Custom options must not be in the XFDL namespace.

Syntax

```
<option xmlns="http://www.PureEdge.com/XFDL/Custom">
   <!-- Arbitrary XML content -->
</option>
```

Example

This sample shows a custom option containing an SQL query.

```
<sql:query
   xmlns:sql="http://www.iso-standards.org/9075/2002/sqlxml">
   <!-- Content describing an SQL query -->
</sql:query>
```

This XML could be included in the definition of an item that triggers a form submission. Since the internal content can use XFDL computes to populate the query based on form content from the user, the server-side processing would be able to perform a proper query and have the results used to populate a response form.

Usage Notes

1. The naming conventions for a custom option are as follows:
   - Custom options can have computed values by using the XFDL compute attribute, which must be qualified with a namespace prefix associated with the XFDL namespace URI.
   - In order to make the XML content addressable by the XFDL compute system, the tag names of the custom option and any elements within it must conform to the XFDL syntax for a scope identifier.

9. Function Calls

XFDL is an assertion-based language, which means a “truth engine” maintains statements in the code as true. The functions described in this section of the specification allow an XFDL form to perform procedural operations that would normally require complicated computations to achieve.

Function calls run code that may be external to the XFDL form definition. Below are the BNF rules for functions.

[45] FunctionCall := (LibName ‘.’)? FunctionName ‘(‘ (Compute (‘,’ Compute)*)? ‘)’

[46] LibName ::= sid

[47] FunctionName ::= sid

The LibName allows functions to be grouped into separate namespaces, but the predefined functions in this specification do not require a LibName. (The LibName assigned to these predefined functions is *system*.) Any user-defined namespace must contain an underscore in its name.

Examples

Calling a predefined function (in the system namespace):

```
<custom:status xfdl:compute="toggle(field1.value, ’high’, ’low’)"
   ></custom:status>
```

or

```
<custom:status xfdl:compute="system.toggle(field1.value, &#xA;
   ’high’, ’low’)"></custom:status>
```

Calling a user-defined function (in a custom namespace)

```
<value compute="hr_funcs.holiday(field1.value, field2.value)"
   ></value>
```

Notes on Using Functions

Position in Strings

The position of the first character in a string is at position zero. For example:

```
This is a string
```

The capital T in the string above is at position zero.

Passing Literals and Option References

An XFDL form will evaluate option references before a function runs, unless the references are surrounded by quotation marks.

- To pass a literal value into a function, surround it in double-quotation marks. For example:

  ```
  <custom:length xfdl:compute="strlen('This is a literal string')"
     ></custom:length>
  ```

- To evaluate an option reference and pass its evaluated value into a function, do not surround the option reference in quotation marks. For example:

  ```
  <custom:length xfdl:compute="strlen(surnameField.value)"
     ></custom:length>
  ```

- To pass an option reference into a function (without evaluating it), surround the option reference in double-quotation marks. For example:

  ```
  <custom:set xfdl:compute="set('statusField.value','Confirmed.')"
     ></custom:set>
  ```

9.1 String Functions

9.1.1 countLines

Counts the number of lines that a string would take up over a given width, and returns the number of lines. The count assumes that the font is a monospaced font, and that the line will be wrapped at the ends of words, and not in the middle of words.

This function is useful if it is necessary to dynamically size items into which a string will be inserted. For example, to insert an entry from a database into a field on a form, dynamically size the height of the field so that all of the text is visible.

Note: *The width must be a character-based width and not a pixel-based width.*

Syntax

```
countLines(string, width)
```

string	*literal string or option reference*	the string to base the measurement on
width	*positive int*	the width, in monospaced characters, to base the measurement on

Returns

The number of lines, or "" (empty) if an error occurs.

Example

In this example, the field's height will be set by the number that *countLines* returns. The calculation is based on a dynamically-generated value, and the field's set width (50).

```
<field sid="commentField">
   <label>Comments</label>
   <itemlocation>
      <ae>
         <ae>below</ae>
         <ae>deptField</ae>
      </ae>
   </itemlocation>
```

```
    <size>
        <ae>50</ae>
        <ae compute="countLines(value, ’50’)"></ae>
    </size>
</field>
```

9.1.2 countWords

Counts the number of words in a specified string.

Syntax

```
countWords(string)
```

string	*literal string or option reference*	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The number of words in the original string, or nothing if an error occurs.

Example

In this example, *countWords* will return the value “5”.

```
<field sid="Field1">
   <label>Test countWords()</label>
   <format>
      <ae>string</ae>
   </format>
   <value compute="countWords('Hello my name is Simon.')"
      ></value>
</field>
```

9.1.3 pad

Pads or truncates an ASCII string to a specified length as explained:

- **Padding** — If the string is shorter than the specified length in the *pad* function then the string is padded with spaces. The pad_orientation parameter determines where the original string is oriented within the characters that make up the new string.
- **Truncating** — If the string is longer than the specified length in the *pad* function, than the string is truncated to the new length and any excess characters are lost. The pad_orientation parameter specifies what part of the original string is saved.

Syntax

```
pad(string, length, pad_orientation)
```

string	*literal string or option reference*	the original string to pad or truncate (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
length	*literal string or option reference*	length of the new string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
pad_orientation	*literal string or option reference*	the position of the original string in the new padded or truncated string. (Enclose literal strings in double quotation marks, do not enclose option references in quotation marks.) This is an optional parameter. The valid choices are **left**, **center**, or **right**. The default value is **left** if the parameter is invalid or is not supplied.

Returns

The string padded or truncated to the specified length, or nothing if an error occurs.

Example

```
<field sid="Field1">
   <label>Test pad(): Center pad</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
```

```
    <value compute="pad('Hello','11','center')"></value>
</field>

<field sid="Field2">
    <label>Test pad(): Right pad</label>
    <format>
        <ae>string</ae>
        <ae>mandatory</ae>
    </format>
    <value compute="pad('Hello','10','right')"></value>
</field>

<field sid="Field3">
    <label>Test pad(): Right Truncate</label>
    <format>
        <ae>string</ae>
        <ae>mandatory</ae>
    </format>
    <value compute="pad('Hello','2','left')"></value>
</field>
```

In Field1 of the previous example, the *pad* function has the pad orientation **center**. Pad inserts 3 spaces on either side of the string “Hello” to create a new string which is “11” characters long and looks like the following:

```
"   Hello   "
```

In Field2 of the previous example, the *pad* function has the pad orientation **right**. Pad inserts 5 spaces at the beginning of the string “Hello” to create a new string which is “10” characters long and looks like the following:

```
"     Hello"
```

In Field3 of the previous example, the *pad* function has the pad orientation **left** and will truncate 4 characters from the end of the string “Hello” to create the new string “He” which is two characters long.

9.1.4 replace

Takes a string and replaces a substring in it (marked by **start** and **end**) with a new string. Returns the resulting string.

If start is less than 0 then the substring will begin on the first character of string. If end is greater than or equal to the length of string then the substring will end on the last character of string. If the new string is not long enough (that is, it does not reach position end), replacement will end with the last character of newString. If the new string is too long (that is, it extends past position end), replacement will end on position end.

An error occurs if start is greater than end, if either of start and end is not a valid integer, or if string is empty.

Syntax

```
replace(string, start, end, newString)
```

string	*literal string or option reference*	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
start	*int*	position of character at the start of the substring (the first character in *string* is zero)
end	*int*	position of character at the end of the substring (the first character in *string* is zero)
newString	*literal string or option reference*	the replacement string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The modified string, or "" (empty) if an error occurs.

Example

In this example, the result of *replace* is “Go east, young man!”.

```
<field sid="replaceField">
   <label>Test replace()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
```

```
    <value>Go west, young man!</value>
    <custom:change xfdl:compute="replace(value, ’3’, ’6’, ’east’)"
        ></custom:change>
</field>
```

9.1.5 strlen

Returns the length of string.

Syntax

```
strlen(string)
```

string	*literal string or option reference*	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

A string containing the length.

Example

In this example, the result of *strlen* is “28”.

```
<field sid="stringLengthField">
   <label>The length of this label is:</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="strlen(label)"></value>
</field>
```

9.1.6 strmatch

Determines if the wildcard string *wild* matches the non-wildcard string *real* and returns the boolean result.

Syntax

```
strmatch(wild, real)
```

wild	*literal string or option reference*	the wildcard string to match (enclose literal strings in double quotation marks, do not enclose option references in quotation marks). Any of the following wild card characters can be used: **?** — represents any one (1) character ***** — represents any number of characters **#** — represents any one (1) numeric character **%** — represents any number of numeric characters **@** — represents any one (1) alphabetic character **!** — represents any number of alphabetic characters (which can include none)
real	*literal string or option reference*	the non-wildcard match string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

A string containing “1” if a match occurs, “0” if no match occurs.

Example

In this example, the result of *strmatch* is “1”.

```
<field sid="testStrmatch">
   <label>Test strmatch()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value>To be or not to be, etc.</value>
   <custom:change xfdl:compute="strmatch(’?o be* ?o be*’, value)"
      ></custom:change>
```

```
</field>
```

9.1.7 strpbrk

Returns the position of the first character in *string1* that matches any of the characters in *string2*. Note that the count is zero based.

Syntax

```
strpbrk(string1, string2)
```

string1	*literal string or option reference*	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
string2	*literal string or option reference*	the string of characters (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

A string containing the position, or “-1” if no matching characters are found.

Example

The result of *strpbrk*, displayed in “FIELD2” in the example below, is “9”.

```
<field sid="testStrpbrk">
   <label>testField</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value>To be or not to be, etc.</value>
   <custom:change xfdl:compute="strpbrk(value, ’lLmMnNOpP’)"
      ></custom:change>
</field>

<field sid="FIELD2">
   <label>result field</label>
   <value compute="testStrpbrk.custom:change"></value>
</field>
```

9.1.8 strrstr

Returns the position of the first character of the last occurrence of *string2* in *string1*.

Syntax

```
strrstr(string1, string2)
```

string1	*literal string or option reference*	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
string2	*literal string or option reference*	the substring (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

A string containing the position, or “-1” if no substring is found.

Example

The result of *strrstr*, displayed in “FIELD2” in the example below, is “16”.

```
<field sid="testStrrstr">
   <label>testField</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value>To be or not to be, etc.</value>
   <custom:change xfdl:compute="strrstr(value, 'be')"
      ></custom:change>
</field>
<field sid="FIELD2">
   <label>result field</label>
   <value compute="testStrrstr.custom:change"></value>
</field>
```

9.1.9 strstr

Returns the position of the first character of the first occurrence of *string2* in *string1*.

Syntax

```
strstr(string1, string2)
```

string1	*literal string or option reference*	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
string2	*literal string or option reference*	the substring (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

A string containing the position, or “-1” if no occurrence is found.

Example

The result of *strstr*, displayed in “FIELD2” in the example below, is “3”.

```
<field sid="testStrstr">
   <label>testField</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
<value>To be or not to be, etc.</value>
   <custom:change xfdl:compute="strstr(value, 'be')"
      ></custom:change>
</field>
<field sid="FIELD2">
   <label>result field</label>
   <value compute="testStrstr.custom:change"></value>
</field>
```

9.1.10 substr

Returns the substring of *string* from the position indicated in *start* through the position indicated in *end*. If *start* is less than zero then the substring will begin on the first character of string. If *end* is greater than or equal to the length of string then the substring will end on the last character of string.

An error occurs if *start* is greater than *end*, if either of *start* and *end* is not a valid integer, or if string is empty.

Syntax

```
substr(string, start, end)
```

string	*literal string or option reference*	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
start	*int*	position of character at the start of the substring (the first character in *string* is zero)
end	*int*	position of character at the end of the substring (the first character in *string* is zero)

Returns

The substring, or "" (empty) if an error occurs.

Example

The result of *substr*, displayed in “FIELD2” in the example below, is “Watso”.

```
<field sid="surnameField">
   <label>Surname</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value>Watson</value>
   <custom:change xfdl:compute="substr(value, ’0’, ’4’)"
      ></custom:change>
</field>
<field sid="FIELD2">
   <label>result field</label>
   <value compute="surnameField.custom:change"></value>
</field>
```

9.1.11 tolower

Returns the lower case of *string*.

Syntax

```
tolower(string)
```

string	*literal string or option reference*	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The lower case string.

Example

The result of *tolower*, shown in “displayField” in the example below, is “hello!”.

```
<field sid="tolowerField">
   <label>Test tolower()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value>Hello!</value>
   <custom:change xfdl:compute="tolower(value)"></custom:change>
</field>
<field sid="displayField">
   <value compute="tolowerField.custom:change"></value>
</field>
```

9.1.12 toupper

Returns the upper case of *string*.

Syntax

```
toupper(string)
```

string	*literal string or option reference*	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The upper case string.

Example

The result of *toupper*, shown in “displayField” in the example below, is “HELLO!”.

```
<field sid="toupperField">
   <label>Test toupper()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value>Hello!</value>
   <custom:change xfdl:compute="toupper(value)"></custom:change>
</field>
<field sid="displayField">
   <value compute="toupperField.custom:change"></value>
</field>
```

9.1.13 trim

Returns a copy of *string* with all leading and trailing white space (blanks, tabs, newlines, carriage returns) removed.

Syntax

```
trim(string)
```

string	*literal string or option reference*	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The string with leading and trailing whitespace removed.

Example

In this example, the result of *trim* is “Test trim()”.

```
<field sid="trimField">
   <label>     Test trim()   </label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="trim(label)"></value>
</field>
```

9.1.14 URLDecode

Returns a URL-decoded version of *string*.

Syntax

```
URLDecode(string)
```

string	*literal string or option reference*	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The URL-decoded string.

Example

In this example, the result of *URLDecode* is “This is a line\r”.

```
<field sid="URLDecodeField">
   <label>Test URLDecode()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="URLDecode('This%20is%20a%20line%OD')"></value>
</field>
```

9.1.15 URLEncode

Returns a URL-encoded version of *string*.

Syntax

```
URLEncode(string)
```

string	*literal string or option reference*	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The URL-encoded string.

Example

In this example, the result of *URLEncode* is “This+is+a+line%0D%0A”.

```
<field sid="URLEncodeField">
   <value compute="URLEncode(’This is a line\n’)"></value>
</field>
```

9.2 Math Functions

9.2.1 abs

Returns the absolute value of the number represented in *number*.

An error occurs if *number* is not a valid number.

Syntax

```
abs(number)
```

number	*decimal number*	a number

Returns

A string containing the absolute of the number, or "" if an error occurs.

Example

In this example, the result of *abs* is “2341.23”.

```
<field sid="absTest">
   <label>Test abs()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="abs(’-2341.23’)"></value>
</field>
```

9.2.2 acos

Returns the arc cosine of a number stored in *number*.

An error occurs if *number* is not a valid number or has absolute value greater than 1.

Syntax

```
acos(number)
```

number	*decimal number*	a number

Returns

A string containing the arc cosine, or "" if an error occurs.

Example

In this example, the result of *acos* is “1.047198”.

```
<field sid="arccosineField">
   <label>Test acos()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="acos(’0.5’)"></value>
</field>
```

9.2.3 annuity

Returns the present value *annuity factor* for an ordinary annuity, at a periodic interest rate indicated by *rate* over a number of periods specified in *periods*. (*Present value* is the lump sum to invest at rate in order to produce a set payment over periods. An *ordinary annuity* provides the payment at the end of each period specified in periods.)

This function might be used to figure out either:

- P, the present value (lump sum to invest).
- R, the periodic payment amount that will be received.

For reference:

- P = R * annuity_factor
- R = P / annuity_factor

An error occurs if *periods* is not a valid integer, or if rate is 0.

Syntax

```
annuity(rate, periods)
```

rate	*decimal number*	the rate of interest in decimal form compounded each period
periods	*integer*	the number of periods

Returns

A string containing the present value annuity factor, or "" if an error occurs.

Example

In this example, annuity returns “5.786373” and, if the desired payment entered into “paymentField” were $1, then the value of “presentValueInv” would be $5.78. (That is, a person would have to invest $5.78 at 5% for seven payments.)

```
<field sid="presentValueInv">
   <label>The present value to invest is:</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="paymentField.value * &#xA;
      annuity(’.05’, ’7’)"></value>
```

```
</field>
```

9.2.4 asin

Returns the arc sine of a number stored in *number*.

An error occurs if *number* is not a valid number or has an absolute value greater than 1.

Syntax

```
asin(number)
```

number	*decimal number*	a number

Returns

A string containing the arc sine, or "" if an error occurs.

Example

In this example, the result of *asin* is “0.523599”.

```
<field sid="arcsinField">
   <label>Test asin()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="asin(’0.5’)"></value>
</field>
```

9.2.5 atan

Returns the arc tangent of a number stored in *number*.

An error occurs if *number* is not a valid number.

Syntax

```
atan(number)
```

number	*decimal number*	a number

Returns

A string containing the arc tangent, or "" if an error occurs.

Example

In this example, the result of *atan* is “0.463648”.

```
<field sid="arctangentField">
   <label>Test atan()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="atan(’0.5’)"></value>
</field>
```

9.2.6 ceiling

Returns the ceiling of the number represented in *number*.

An error occurs if *number* is not a valid number.

Syntax

```
ceiling(number)
```

number	*decimal number*	a number

Returns

A string containing the ceiling of the number, or "" if an error occurs.

Example

In this example, the result of *ceiling* is “-19”.

```
<field sid="ceilingTest">
   <label>Test ceiling()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="ceiling(’-19.6’)"></value>
</field>
```

9.2.7 compound

Returns the compound interest factor at a rate indicated by *rate* over a number of periods specified in *periods*.

This might be used to calculate the total amount of a loan, by multiplying an original principle by the result of *compound*. See below for an example.

An error occurs if *periods* is not a valid integer.

Syntax

```
compound(rate, periods)
```

rate	*decimal number*	the rate of interest in decimal form compounded each period
periods	*integer*	the number of periods

Return

A string containing the compound interest factor, or "" if an error occurs.

Example

In this example, the result of *compound* is “1.948717”. The value of the field is 1.948717 x the amount in the “principleField”.

```
<field sid="totalAmountField">
   <label>Total Amount of Loan</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="principleField.value * &#xA;
      compound(’.1’, ’7’)"></value>
</field>
```

9.2.8 cos

Returns the cosine of an angle stored in *angle* and expressed in radians.

An error occurs if *angle* does not contain a valid angle.

Syntax

```
cos(angle)
```

angle	*decimal number*	the angle in radians

Returns

A string containing the cosine, or "" if an error occurs.

Example

In this example, the result of *cos* is “-0.416147”.

```
<field sid="cosineField">
   <label>Test cos()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="cos(’2’)"></value>
</field>
```

9.2.9 deg2rad

Returns the number of radians in an angle expressed in degrees stored in *angle*.

An error occurs if *angle* does not contain a valid angle.

Syntax

```
deg2rad(angle)
```

angle	*decimal number*	the angle in degrees

Returns

A string containing the number of radians, or "" if an error occurs.

Example

In this example, the result of *deg2rad* is “2.00000”.

```
<field sid="deg2radField">
   <label>Test deg2rad()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="deg2rad(’114.591559’)"></value>
</field>
```

9.2.10 exp

Returns the exponentiation of the number represented in *number* (i.e., e^{number}).

An error occurs if *number* is not a valid number.

Syntax

```
exp(number)
```

number	*decimal number*	a number

Returns

A string containing the exponentiation of the number, or "" if an error occurs.

Example

In this example, the result of *exp* is “20.855369”.

```
<field sid="expTestField">
   <label>Test exp()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="exp(’3’)"></value>
</field>
```

9.2.11 fact

Returns the factorial value of the integer represented in *integer*.

An error occurs if *integer* is negative.

Syntax

```
fact(number)
```

integer	*integer*	a non-negative integer

Returns

A string containing the factorial of the integer, or "" if an error occurs.

Example

In this example, the result of *fact* is “40320”.

```
<field sid="factTestField">
   <label>Test fact()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="fact(’8’)"></value>
</field>
```

9.2.12 floor

Returns the floor of the number represented in *number*.

An error occurs if *number* is not a valid number.

Syntax

```
floor(number)
```

number	*decimal number*	a number

Returns

A string containing the floor of the number, or "" if an error occurs.

Example

In this example, the result of *floor* is “-20”.

```
<field sid="floorTestField">
   <label>Test floor()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="floor(’-19.6’)"></value>
</field>
```

9.2.13 ln

Returns the natural logarithm of the number represented in *number*.

An error occurs if *number* is not a decimal number greater than zero.

Syntax

```
ln(number)
```

number	*decimal number*	a number

Returns

A string containing the natural log of the number, or "" if an error occurs.

Example

In this example, the result of *ln* is “0”.

```
<field sid="lnTestField">
   <label>Test ln()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="ln(’1’)"></value>
</field>
```

9.2.14 log

Returns the logarithm of the number represented in *number* to the base indicated by *base*.

An error occurs if either of *number* or *base* is not a valid number. The *number* must be equal to, or greater than 1.

Syntax

```
log(number, base)
```

number	*decimal number*	a number
base	*decimal number*	optional. A number representing the base for which the logarithm will be computed. If no base is supplied, a base of 10 is used.

Returns

A string containing the log of the number to the base, or "" if an error occurs.

Example

In this example, the result of *log* is “2”.

```
<field sid="logTestField">
   <label>Test log()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="log(’100’, ’10’)"></value>
</field>
```

9.2.15 mod

Returns the modulus of the number represented in *number* using the divisor indicated by *divisor*.

An error occurs if either of *number* or *divisor* is not a valid number, or *divisor* is 0.

Syntax

```
mod(number, divisor)
```

number	*decimal number*	a number
divisor	*decimal number*	a number representing the divisor for which the modulus will be computed

Returns

A string containing the modulus, or "" if an error occurs.

Example

In this example, the result of *mod* is “-0.200000”.

```
<field sid="modTestField">
   <label>Test mod()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="mod(’-3.5’, ’.3’)"></value>
</field>
```

9.2.16 pi

Returns the value of PI to 13 decimal places.

Syntax

```
pi()
```

Returns

A string containing the value of π.

Example

In this example, the result of *pi* is “3.14159265359” (precision is software-dependent).

```
<field sid="piTestField">
   <label>Test pi()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="pi()"></value>
</field>
```

9.2.17 power

Returns the number represented in *number* raised to the power indicated by *power.*

An error occurs if either of *number* or *power* is not a valid number.

Syntax

```
power(number, power)
```

number	*decimal number*	a number
power	*decimal number*	a number representing the power by which the number will be raised

Returns

A string containing the number raised to the power, or "" if an error occurs.

Example

In this example, the result of *power* is “100.00000”.

```
<field sid="powerTestField">
   <label>Test power()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="power(’0.1’, ’-2’)"></value>
</field>
```

9.2.18 rad2deg

Returns the number of degrees in an angle expressed in radians stored in *angle*.

An error occurs if *angle* does not contain a valid angle.

Syntax

```
rad2deg(angle)
```

angle	*decimal number*	the angle in radians

Returns

A string containing the number of degrees, or "" if an error occurs.

Example

In this example, the result of *rad2deg* is “114.591559”.

```
<field sid="rad2degField">
   <label>Test rad2deg()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="rad2deg(’2’)"></value>
</field>
```

9.2.19 rand

Returns a random integer from the range of integers indicated by *lowerlimit* and *upperlimit*. (The range includes *lowerlimit* and *upperlimit*).

An error occurs if either of *lowerlimit* or *upperlimit* is not a valid integer, or *upperlimit* is less than *lowerlimit*.

Syntax

```
rand(lowerlimit, upperlimit)
```

lowerlimit	*integer*	the lower limit of the random number's range
upperlimit	*integer*	the upper limit of the random number's range

Returns

A string containing the random integer, or "" if an error occurs.

Example

In this example, the result of *rand* is an integer in the range [45,90].

```
<field sid="randTestField">
   <label>Test rand()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="rand('45', '90')"></value>
</field>
```

9.2.20 round

Returns the number represented in *number* rounded to the nearest decimal position indicated by *place* (e.g., 100, 10, 1, 0.1, ...). All numbers rounded to right of an *x/y* graph, so the result of rounding negative numbers goes the opposite way you would expect. See below for an example.

An error occurs if *number* is not a valid number or *place* is not a power of 10.

Syntax

```
round(number, place)
```

number	*decimal number*	a number
place	*decimal number*	a number representing the decimal place where number is to be rounded

Returns

A string containing the rounded number, or "" if an error occurs.

Examples

In this example, the result of *round* is “323.2400”.

```
<field sid="roundTestField">
   <label>Test round()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="round(’323.235’, ’.01’)"></value>
</field>
```

In this example, the result of *round* is “-323.2300”.

```
<field sid="roundTestField">
   <label>Test round()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="round(’-323.235’, ’.01’)"></value>
</field>
```

9.2.21 sin

Returns the sine of an angle stored in *angle* and expressed in radians.

An error occurs if *angle* does not contain a valid angle.

Syntax

```
sin(angle)
```

angle	*decimal number*	the angle in radians

Returns

A string containing the sine, or "" if an error occurs.

Example

In this example, the result of *sin* is “0.909297”.

```
<field sid="sineField">
   <label>Test sin()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="sin(’2’)"></value>
</field>
```

9.2.22 sqrt

Returns the square root of the number represented in *number*.

An error occurs if *number* is a negative number.

Syntax

```
sqrt(number)
```

number	*decimal number*	a non-negative number

Returns

A string containing the square root, or "" if an error occurs.

Example

In this example, the result of *sqrt* is “4.415880”.

```
<field sid="sqrtTestField">
   <label>Test sqrt()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="sqrt(’19.5’)"></value>
</field>
```

9.2.23 tan

Returns the tangent of an angle expressed in radians stored in *angle*.

An error occurs if *angle* does not contain a valid angle (for example, π/2, 3π/2, 5π/2, and so on).

Syntax

```
tan(angle)
```

angle	*decimal number*	the angle in radians

Returns

A string containing the tangent, or "" if an error occurs.

Example

In this example, the result of *tan* is “-2.185040”.

```
<field sid="tanField">
   <label>Test tan()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="tan(’2’)"></value>
</field>
```

9.3 Utility Functions

9.3.1 applicationName

Returns the name of the currently running application.

Syntax

```
applicationName()
```

Returns

A string containing the application name.

Example

In this example, if the form were running in an application named “Viewer”, the result of *applicationName* would be “Viewer”.

```
<field sid="appNameField">
   <label>Test applicationName()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="applicationName()"></value>
</field>
```

9.3.2 applicationVersion

Returns the version of the currently running application in the format “MM.mm.TT”, where MM is the Major version number, mm is the minor version number, TT is the maintenance number.

Syntax

```
applicationVersion()
```

Returns

A string containing the application version.

Example

In this example, if running in an application of version 3.2.4, the result of *applicationVersion* would be “03.02.04”.

```
<field sid="appVersionField">
   <label>Test applicationVersion()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="applicationVersion()"></value>
</field>
```

9.3.3 applicationVersionNum

Returns the decimal form of the version of the currently running application. This number is obtained from the hexadecimal format 0xMMmmTTPP, where MM is the Major version number, mm is the minor version number, TT is the maintenance number, and PP is the patch number. At this point, individual patches are not recognized in version numbers and so will always be 0.

Syntax

```
applicationVersionNum()
```

Returns

A string containing the application version number.

Example

In this example, if running in an application at version v3.2.4, the result of *applicationVersionNum* would be “50463744”, which is the decimal representation of 0x03020400.

```
<field sid="appVersionNumField">
   <label>Test applicationVersionNum()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="applicationVersionNum()"></value>
</field>
```

9.3.4 checkValidFormats

This function checks the format of *all* items in the form and returns the number of items whose format is invalid. If the function call includes an optional reference parameter, the function also provides references to the invalid items.

To check the validity of a single item, use the *isValidFormat* function (see section “9.3.18 isValidFormat”).

Syntax

```
CheckValidFormats(reference)
```

reference	*string*	optional. When present, it refers to an option containing references to each item that failed the validity check. If this option does not exist, the function creates it. If the option already exists, the function replaces its contents with the new list of invalid items. This option is created in the same item that the function is called from. If this parameter is not specified in the call, the function still validates all form items but does not create a list of references to invalid items.

Returns

Returns the number of items that failed the validity check. If there are no invalid items, the function returns 0.

Example

The following sample form shows how you can use the *checkValidFormats* function to specify custom behaviors if a form with invalid values is submitted:

```
<?xml version="1.0"?>
<XFDL xmlns="http://www.PureEdge.com/XFDL/6.0"
   xmlns:xfdl="http://www.PureEdge.com/XFDL/6.0">
   <globalpage sid="global">
      <global sid="global"></global>
   </globalpage>
<page sid="PAGE1">
   <global sid="global"></global>
   <field sid="FIELD1">
      <format>
```

```
            <ae>date</ae>
            <ae>mandatory</ae>
        </format>
    </field>
    <field sid="FIELD2">
        <value>4.00</value>
        <format>
            <ae>dollar</ae>
            <ae>mandatory</ae>
        </format>
    </field>
    <button sid="BUTTON1">
        <value>Submit</value>
        <type>done</type>
        <url>
            <ae>http://localhost/cgi-bin/test.pl</ae>
        </url>
        <checkIfValid compute=
            "toggle(BUTTON1.activated, 'off', 'on') == '1' ? &#xA;
            (checkValidFormats('custom:brokenOptions') != '0' ? &#xA;
            set('BUTTON1.activated', 'off') + &#xA;
            messageBox('Unable to send because at least ' &#xA;
            +. brokenOptions[0] +. ' item is invalid.') : '') : &#xA;
            ''"></checkIfValid>
    </button>
</page>
</XFDL>
```

The *format* option for “FIELD1” specifies that this item must contain a date and cannot be blank. If a user submits the form without correctly completing this field, the *checkValidFormats* function prevents the submission. Instead, the function creates a new option called “brokenOptions”:

```
<custom:brokenOptions>
    <ae>PAGE1.FIELD1</ae>
</custom:brokenOptions>
```

The remaining code uses the information in brokenOptions to display a context-sensitive message to the user.

9.3.5 countDatagroupItems

Returns the number of items in a particular datagroup.

Syntax

```
countDatagroupItems(datagroup)
```

datagroup	*string*	the name of the datagroup. This can include a page reference, such as *Page1.myGroup*. If it does not, the function searches for the group on the page that contains the function.

Returns

The number of items in the datagroup or "" if an error occurs.

Example

In this example, the field displays the number of items in a datagroup called *Data1*.

```
<field sid="totalCount">
   <label>The Green group contains this many items:</label>
   <value compute="countDatagroupItems("Data1")"></value>
</field>
```

9.3.6 countGroupedItems

Returns any of the following:

- The total number of items in a group.
- The total number of items in a group that have a particular option. For example, the number of items with a *bgcolor* setting in a group.
- The total number of items in a group that have a particular option setting. For example, the number of items with a *bgcolor* set to *blue* in a group.

Syntax

```
countGroupedItems(group, option, literal, groupContext,
groupContextType, referenceType, scheme)
```

group	*string*	the name of the group you want to get an item from. This can include a page reference, such as *Page1.myGroup*.
option	*string*	*optional*. If supplied, the function will only count items that contain this option.
literal	*string*	*optional*. If supplied, the function will only count items that have the specified option set to this value. Must be used with the *option* parameter.
groupContext	*string*	*optional*. The starting point to use to locate the group if the group name is not fully qualified. For example, if the group was on the first page, you would use *Page1*.
groupContext Type	*string*	*optional*. The level of the group context parameter, such as *page*, *item*, or *option*. Currently only *page* is valid.
referenceType	*string*	*optional*. Sets the scope of the reference that is returned. The reference begins at the level below this. For example, to get a reference that begins at the page level, set this parameter to *form*. Valid settings are *form* and *page*. Default is *form*.
scheme	*string*	*optional*. The referencing scheme used. Default is *XFDL*.

Returns

The number of items that match the search criteria or "" if an error occurs.

Example

In this example, the field displays the number of items in a group called *Green*.

```
<field sid="totalCount">
   <label>The Green group contains this many items:</label>
   <value compute="countGroupedItems("Page1.Green")"></value>
</field>
```

9.3.7 decimal

Returns the decimal representation of the number represented by *number* with base indicated by *base*.

An error occurs if *number* is not a valid number, if *base* is not a valid positive integer base, or *number* cannot be resolved under the specified *base*.

Syntax

```
decimal(number, base)
```

number	*number*	a number
base	*positive integer*	an integer that is the base of the provided number

Returns

A string containing the decimal representation of the number, or "" if an error occurs.

Example

In this example, the result of *decimal* is “-74”.

```
<field sid="decimalTestField">
   <label>Test decimal()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="decimal(’-4A’, ’16’)"></value>
</field>
```

9.3.8 destroy

Destroys (or deletes) specified elements from a form, including boxes, buttons, checks, comboboxes, fields, labels, lines, lists, pages, popups, radios, and spacers.

When a compute within the *destroy* function causes a change to the form, the form checks for other computes referenced by the *destroy* function and evaluates them immediately.

Syntax

```
destroy(reference, type)
```

reference	*string*	a reference to the element you want to destroy
type	*string*	the type of element being destroyed; types can be page, item, or option

Returns

Nothing if the operation was successful, or an error message if the reference cannot be destroyed.

Example

This example creates a button that, when clicked by the user, deletes “FIELD1” from “PAGE2” of the form.

```
<button sid="deleteField_BUTTON">
   <value>Delete</value>
   <custom:destroy xfdl:compute=
      "toggle(activated, ’off’, ’on’) == ’1’ ? &#xA;
      destroy(’PAGE2.FIELD1’, ’item’) + &#xA;
      set(’global.global.dirtyflag’, ’on’) : ’’"
      ></custom:destroy>
</button>
```

9.3.9 duplicate

Makes a copy of a specified form element, places the copy in a designated location, and assigns it a new name. Computes in duplicate elements are evaluated immediately.

When creating elements for duplication, you will need to use relative tags (see section “3.7 Function Call Syntax”). Also, see the *generateUniqueName* and *getReference* functions.

Syntax

```
duplicate(reference, type, newReference, newType,
location, newName)
```

reference	*string*	a reference to the element you want to duplicate. For example, if you want to duplicate “FIELD1” on “PAGE1”, the reference would be *PAGE1.FIELD1*.
type	*string*	the type of element being duplicated. Types can be page, item or option. For example, if you want to duplicate an element whose reference was *PAGE1.FIELD1*, the type would be item.
newReference	*string*	a reference that identifies where to put the new element. The new element is created as either a child or sibling of this element. For example, to create an element as a sibling of *FIELD1*, you would use: *PAGE1.FIELD1*.
newType	*string*	the type of the newReference. Types can be page, item, or option. For example, if the newReference is *PAGE2*, the newType would be page; if the newReference is *FIELD8*, the newType would be item.
location	*string*	a description of where to put the new element in relation to the newReference supplied in newType. Valid settings are: ■ **append_child** — adds the new element as the last child of the newType. ■ **after_sibling** — adds the new element as a sibling of the newType, placing it immediately after that node in the form structure. ■ **before_sibling** — adds the new element as a sibling of the newType, placing it immediately before that node in the form structure.

newName	*string*	a new name for the duplicate element

Returns

The duplicate element containing the original element settings.

Example

In this example, the option custom:duplicate calls the *duplicate* function to make a copy of “Field1” on “Page1”, which is identified as an *item* type. The call then places the duplicate on “Page2”, which is identified as a *page* type, and places it immediately after the last item on “Page2” (**append_child**).

```
<button sid="duplicateFieldButton">
   <custom:duplicate xfdl:compute=
      "toggle(activated, ’off’, ’on’) == ’1’ ? &#xA;
      duplicate(’Page1.Field1’, ’item’, ’Page2’, ’page’, &#xA;
      ’append_child’, new_Name) : ’’"></custom:duplicate>
</button>
```

9.3.10 formatString

Returns a string formatted according to the rules set out in a referenced *format* option.

An error occurs if an invalid *format* is specified.

Syntax

```
formatString(reference, formatOptionReference)
```

reference	*string*	a reference to the string to reformat. For example, to format the value contained in “Field3”, the reference would be *Field3.value*.
formatOption Reference	*string*	the format option used to apply to the reference. For example, to format the reference with the format options defined in “Field1”, the string would be *Field1.format*.

Returns

The formatted string.

Example

In this example, the result of *formatString* in “Field2” is “$30,095.60”.

```
<field sid="Field1">
   <label>Field 1</label>
   <format>
      <ae>dollar</ae>
      <ae>add_ds</ae>
      <ae>comma_delimit</ae>
   </format>
   <value></value>
</field>
<field sid="Field2">
   <label>Field 2</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="formatString(Field3.value, ’Field1.format’)"
      ></value>
</field>
<field sid="Field3">
   <label>Field 3</label>
```

```
    <value>30095.6</value>
</field>
```

In this example, *formatString* reformats a value as an integer and inserts it into *custom:value*.

```
<field sid="Field4">
    <value>$1.00</value>
    <format>
        <ae>dollar</ae>
    </format>
    <custom:value xfdl:compute="formatString(value,
        'custom:format')"></custom:value>
    <custom:format>
        <ae>integer</ae>
    </custom:format>
</field>
```

9.3.11 generateUniqueName

Creates a unique name for an element, and is usually used with the *duplicate* function. This name will consist of a name prefix and an integer. The function will search the form for any pages, items or options with the same name as the specified prefix.

For example, consider a button that contains the code to copy a field (using the *duplicate* function). You could include the *generateUniqueName* function in this code. As a result, when a user clicks this button, the newly created field would have a unique name (such as “newField_1”), which is composed of a designated prefix (“newField_”) and an automatically assigned integer (beginning at “1”).

If the name already exists within the scope of the parent element, then the function will increment the integer until a unique name can be generated.

When generating unique names using this function, you will need to use the *set* function.

Syntax

```
generateUniqueName(reference, type, prefix)
```

reference	*string*	a reference to the parent element of the element for which you wish to generate a unique name. For example, if you want to generate a unique name for an item on “PAGE1”, the reference would be *PAGE1*. To indicate that the parent element is the current form, leave this parameter blank. You would do this if you wanted to generate a unique name for a page.
type	*string*	the type of node identified in the reference. For example, if the reference is *PAGE1*, the type would be “page”.
prefix	*string*	the prefix for the unique name generated. This can be any string you like. For example, if you choose “newField_” as the prefix, the first name generated by the function would be “newField_1”, the second would be “newField_2”, and so on.

Returns

A string that is composed of characters and an integer.

Example

In this example, the *generateUniqueName* function generates a unique name and returns the result newField_1.

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

9.3.12 get

Returns the value of a particular form option.

Syntax

```
get(reference, scheme)
```

reference	*string*	a reference to the string to retrieve, or get. For example, to retrieve the value of “Field1”, the reference would be *Field1.value*.
scheme	*string*	*optional*. The referencing scheme used. Defaults to *XFDL*.

Returns

The value of the form option reference or "" if an error occurs.

Example

In this example, *get* retrieves the value of “Field2”, which is “gold”.

```
<field sid="Field1">
   <label>Field 1</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value>gold</value>
</field>
<field sid="Field2">
   <label>Test get()</label>
   <format>
      <ae>string</ae>
   </format>
   <value compute="get(’Field1.value’)"></value>
</field>
```

9.3.13 getAttr

Returns the value of an attribute on a form element.

Syntax

```
getAttr(reference, type, attrName, scheme)
```

reference	*string*	a reference to the element that has the attribute. For example, to get an attribute from a custom data element in *Field1*, you might use: `Page1.Field1.custom:data` All namespace prefixes are resolved relative to this node.
type	*string*	the type of reference used. This is one of *page*, *item*, *option*, or *array.*
attrName	*string*	the name of the attribute, including the appropriate namespace. For example: `prefix:attribute` Use *null* for the empty namespace, or nothing for the XFDL namespace. All other namespace prefixes are resolved relative to the *reference* node supplied.
scheme	*string*	*optional*. The referencing scheme used. Defaults to *XFDL*.

Returns

The value of the attribute or "" if an error occurs.

Example

In this example, *getAttr* retrieves the value of the *id* attribute from the <custom:data> element.

```
<field sid="nameField">
   <value>John B.</value>
   <custom:data id="12"></custom:data>
</field>
<field sid="idField">
   <value compute="getAttr(’nameField.custom:data’, ’option’,
```

```
        'id')"></value>
</field>
```

9.3.14 getGroupedItem

Returns a reference to any of the following:

- The first item in a given group.
- The first item in a group that has a particular option. For example, the first item with a *bgcolor*.
- The first item in a group that has a particular option setting. For example, the first item with a *value* of **on**.

Syntax

```
getGroupedItem(group, option, literal, groupContext,
groupContextType, itemScopeType, scheme)
```

group	*string*	the name of the group you want to get an item from. This can include a page reference, such as *Page1.myGroup*.
option	*string*	*optional*. If supplied, the function will search for an item that has this option.
literal	*string*	*optional*. If supplied, the function will search for an item that has the specified option set to this value. Must be used with the *option* parameter.
groupContext	*string*	*optional*. The starting point to use to locate the group if the group name is not fully qualified. For example, if the group was on the first page, you would use *Page1*.
groupContext Type	*string*	*optional*. The level of the group context parameter, such as *page*, *item*, or *option*. Currently only *page* is valid.
referenceType	*string*	*optional*. Sets the scope of the reference that is returned. The reference begins at the level below this. For example, to get a reference that begins at the page level, set this parameter to *form*. Valid settings are *form* and *page*. Default is *form*.
scheme	*string*	*optional*. The referencing scheme used. Default is *XFDL*.

Returns

A reference to the first item matching the search criteria or "" if an error occurs.

Example

In this example, *getGroupedItem* gets the sid of the radio button that is turned on.

```
<field sid="Field1">
   <label>The SID of the selected radio button is:</label>
   <value compute="getGroupedItem('Page1.radioGroup', 'value',
      'on')"></value>
</field>
```

9.3.15 getPosition

Returns the position index for an element within its parent.

For example, if a page contained two fields, and you called this function on the second field, it would return a value of “1”, indicating that it was the second child (indexing is zero based).

Syntax

```
getPosition(reference, type, scheme)
```

reference	*string*	a reference to the element whose position you want to determine.
type	*string*	the type of reference used. This is one of *page*, *item*, *option*, or *array*.
scheme	*string*	*optional*. The referencing scheme used. Defaults to *XFDL*.

Returns

An integer representing the position of the element within its parent or "" if an error occurs.

Example

This example uses *getPosition* to determine the position index of the *lastNameField* on the first page. It then adds one to the index to change change it from zero based indexing to one based indexing, and concatenates that value into a string that reads: “The lastNameField is element #2 on page one.”

```
<page sid="Page1">
   <field sid="firstNameField"></field>
   <field sid="lastNameField"></field>
</page>
<page sid="Page2">
   <field sid="positionField">
      <value compute="’The lastNameField is element #’ +. &#xA;
         (getPosition(’Page1.lastNameField’, ’item’) + 1) &#xA;
         +. ’ on page one.’"></value>
   </field>
</page>
```

9.3.16 getPref

This returns the value of any setting in the Viewer Preferences form. For example, you could use this function to retrieve the user's email address from the Preferences form.

Syntax

```
getPref(prefName)
```

prefName	*string*	the name of the Preferences setting to get. See the *Notes* below for a list of valid names.

Returns

A string containing the value of the Preferences setting or "" if an error occurs.

Notes

The following table lists shows the names for each setting in the Preferences form. Unless otherwise noted, each setting will have a value appropriate to the *value* option of the item type listed. For example, a field can be set to any string value, a check box can be set to *on* or *off*, and so on.

Preferences Type	Setting	Name
Basic	Network Access — Popup	networkAccess
	Try to locate browser automatically — Check Box	locateBrowser
	Path to Browser — Field	overrideDefaultPathTo Browser
	Use Default Simple MAPI Client — Check Box	useMAPI
	SMTP Server — Field	mailHost
	Return Address — Field	returnEmailAddress
	Use Screen Reader — Check Box	
	Use Enhanced Focus Indicator — Check Box	focusIndicator
Input	Do Predictive Input Checking — Check Box	predictiveChecking

Preferences Type	Setting	Name
	I Prefer to Enter Dates — Radio Buttons Valid values are: *DDMMYY*, *MMDDYY*, and *YYMMDD*.	defaultDateFormat
	Stop Tab From Invalid Input Items — Check Box	forbidTaboutOnError
	Stop Tab From Empty Mandatory Items — Check Box	forbidTaboutOnMandatory
Printing	Print Radio Buttons as Check Boxes — Check Box	printRadiosAsChecks
	Print Radio Buttons Without Values — Check Box	printRadiosWithoutValues
	Print Scroll Bars on Fields — Check Box	printScrollbarsOnFields
	Print Single Line Fields as Lines — Check Box	printSingleLineFieldsAs Lines
	Print Border Around Form Edge — Check Box	printFormBorder
	Page Layout Defaults — Radio Buttons Valid values are: *fitToPage*, *shrinkToPage*, *tileOneDirection*, and *tileTwoDirections*.	printedPageLayout
	Print each page as separate print job — Check Box	printPageSeparately
	Print black and white (excluding images) — Check Box	printBlackAndWhite
Advanced	Show Boundary on All Form Items — Check Box	displayBoundingBoxes
	Use ‘X’ Style Check Boxes — Check Box Valid values are: *X* and *check*.	checkBoxStyle
	Digital Certificate Identity Filter — Field	certIdentifyFilter
	Path to Netscape Profile — Field	overrideDefaultPathTo NetscapeProfile
	Override Netscape Sockets	overrideNetscapeSockets
	Override Internet Explorer Sockets	overrideIESockets
	Check CRL Distribution Points	checkCRLDistribution Points

Example

In this example, *getPref* is used to automatically populate a field with the user's email address:

```
<field sid="emailAddress">
   <label>Email address:</label>
   <value compute="getPref('returnEmailAddress')"></value>
</field>
```

9.3.17 getReference

Returns a reference for the element that contains the call. The function works as a “Where am I?” check for a page, item, or option. It is always called from within the element for which the reference is needed, but the returned reference can be for that element or any of its parents.

For example, *getReference* could be called from within a page’s label, and return a reference for that option (for example, *PAGE1.global.label*), its parent item (for example, *PAGE1.global*), or the page it is on (for example, *PAGE1*).

This function is especially useful when duplicating pages, since new pages will contain the identical options and items as their originals.

Syntax

```
getReference(reference, type, level, scheme)
```

reference	*string*	the element you wish to identify. Possible values include the page sid, the item, or option type, and the name or value of the item or option. For instance, to return the reference for a *label* option containing the call, the reference would be “label”.
type	*string*	the reference type, which can be **page**, **item**, **option**, or **array**. For example, the reference type of a label would be **option**.
level	*string*	the level in the reference for the function to return. If the function is called from an option, but level is identified as *item*, the reference returned would be *page.item*.
scheme	*string*	*optional*. The referencing scheme used. Defaults to *XFDL*.

Returns

A reference to the element (page, item, or option) from which the function was called, or "" if an error occurs.

Example

In this example, *getReference* returns a reference to the page sid: “PAGE1”.

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

```
        <label compute="getReference(’label’, ’option’, ’page’)"
            ></label>
    </global>
```

In this example, *getReference* returns a reference to the option of *label*, which in this case is *PAGE1.global.label*.

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

9.3.18 isValidFormat

Returns the boolean result of whether a string is valid according to the setting of the format option referred to in *formatOptionReference*.

An error occurs if a non-existent format is specified.

Syntax

```
isValidFormat(string, formatOptionReference)
```

string	*string*	a string to be checked against the format. For example, to check 23.2 against a specific format, the string would be “23.2”.
formatOption Reference	*string*	the option reference of the format, including the page tag if necessary, to check the string against. For example, to check 23.2 against a format specified in Field1, the formatOptionReference would be “Field1.format”.

Returns

“1” if the string follows the format, “0” if not, or "" if an error occurs.

Example

In this example, the result of *isValidFormat* is “0” because the string to check contains a non-integer number representation and the specified format to check is of type integer.

```
<field sid="Field1">
   <label>Field 1</label>
   <format>
      <ae>integer</ae>
      <ae>mandatory</ae>
   </format>
   <value>45</value>
</field>
<field sid="Field2">
   <label>Test isValidFormat()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="isValidFormat(’23.2’, ’Field1.format’)"
      ></value>
```

```
</field>
```

9.3.19 set

Sets the value of a form option described by reference to the value described by value and returns an indication of the success of the operation. The option will be created if it does not exist. If a compute existed on the option, it will be destroyed. Items and pages will not be created.

You can also use the *set* function to reference radio buttons in computes for other options. For example, if a form user selects one check box (thereby turning its value **on**), the *set* function can turn the value of another check box **off**. If you use a *set* function to turn radio button **on** (or **off**), you must also use a set function to turn each of the other radio buttons **off** (or **on**).

When a compute within the *set* function causes a change to the form, the form checks for other computes referenced by the *set* function and evaluates them immediately.

An error occurs if the specified form option could not be set to the specified value.

Syntax

```
set(reference, value, scheme)
```

reference	*string*	a reference to the option to set. For example, to set the value of “Field1”, the reference would be *Field1.value*; to set the value of a check box, the reference would be *CHECK2.value*.
value	*string*	the reference’s new value. For example, to set the reference’s new value to **silver**, the value would be silver; to set the value of a checkmark to **off**, the value would be **off**.
scheme	*string*	*optional*. The referencing scheme used. Defaults to *XFDL*.

Returns

“1” if the operation completed successfully, “0” if an error occurred.

Example

In this example, the result of *set* is “1” and the value of Field1 is set to “silver”.

```
<field sid="Field1">
   <label>Field 1</label>
   <value>gold</value>
</field>
<field sid="Field2">
```

```
    <label>Test set()</label>
    <value compute="set('Field1.value', 'silver')"></value>
</field>
```

In this example, if a form user selects the “CHECK1” check box (thereby turning its value **on**), the *set* function sets the value of “CHECK2” to **off**.

```
<check sid="CHECK1">
    <custom:set xfdl:compute="value == 'on' ? &#xA;
        set('CHECK2.value', 'off') : ''"></custom:set>
    <value>off</value>
</check>
<check sid="CHECK2">
    <custom:set xfdl:compute="value == 'on' ? &#xA;
        set('CHECK1.value', 'off') : ''"></custom:set>
    <value>off</value>
</check>
```

9.3.20 setAttr

Sets the value of an attribute on a form element.

Syntax

```
setAttr(reference, type, attrName, value, scheme)
```

reference	*string*	a reference to the element that has the attribute. For example, to set an attribute from a custom data element in *Field1*, you might use: `Page1.Field1.custom:data` All other namespace prefixes are resolved relative to this node.
type	*string*	the type of reference used. This is one of *page*, *item*, *option*, or *array.*
attrName	*string*	the name of the attribute, including the appropriate namespace. For example: `prefix:attribute` Use *null* for the empty namespace, or nothing for the XFDL namespace. All other namespace prefixes are resolved relative to the *reference* node supplied.
value	*string*	the value to assign to the attribute.
scheme	*string*	*optional*. The referencing scheme used. Defaults to *XFDL*.

Returns

“1” if the operation completed successfully or “0” if an error occurred.

Example

In this example, *setAttr* sets the value of the *id* attribute in the <custom:data> element to 12.

```
<field sid="nameField">
   <value>John B.</value>
   <custom:data id=""></custom:data>
</field>
```

```
<field sid="idField">
   <value compute="setAttr(’nameField.custom:data’, ’option’,
      ’id’, ’12’)"></value>
</field>
```

9.3.21 toggle

Monitors a specific form option and detects any changes to the value of that option. This function can detect any change in an option, or can watch for specific changes. For example, you can create a toggle that detects when an option changes from **on** to **off**, or from **Fred** to **Jim**.

An error occurs if the specified form option does not exist.

Syntax

```
toggle(reference, start, end)
```

reference	*string*	a reference to the option to watch. For example, to watch the value of a check box called “noChoiceAllowed”, the reference would be *noChoiceAllowed.value*.
start	*string*	*optional*. The start condition for toggle. For example, you would set this to **off** to monitor when a check box is checked (the check box will toggle from **off** to **on** when it is checked by the user).
end	*string*	*optional*. The end condition for toggle. For example, you would set this to **on** to monitor when a check box is checked (the check box will toggle from **off** to **on** when it is checked by the user).

Returns

“1” if the specified change occurs in the specified option, or “0” if another change occurs.

Example

In this example, *toggle* monitors a specific option for any change. Every time the value of “nameField” changes, toggle will return “1”, and then a new time will be entered into “timeStampField”, using the *now* function.

```
<field sid="timeStampField">
   <value compute="toggle(nameField.value) == '1' ? now() : ''"
      ></value>
   <label>Time Stamp</label>
   <editstate>readonly</editstate>
</field>
<field sid="nameField">
   <label>Name</label>
   <value></value>
```

```
</field>
```

In this example, *toggle* monitors a check box to determine if the check box is checked. If the value of the check box goes from off to on, the value of the label will change to “The box has been checked.”

```
<label sid="checkStatusLabel">
   <value compute="toggle(check1.value, ’off’, ’on’) == ’1’ ? &#xA;
      ’The box has been checked.’ : &#xA;
      ’The box has not been checked.’"></value>
</label>
<check sid="check1">
   <value>off</value>
</check>
```

9.3.22 xmlmodelUpdate

This function updates the XML data model in the form. This is useful if computes have changed the structure of the data model in some way, such as changing or adding bindings. These sorts of changes do not take effect until the *xmlmodelUpdate* function is called.

Syntax

```
xmlmodelUpdate()
```

Returns

“1” if the operation completed successfully or “0” if an error occurred.

Example

The following XML data model has two instances for customer data. In this case, the second data instance is bound to the form, linking first name and last name.

```
<xmlmodel>
   <instances>
      <xforms:instance xmlns="http://www.w3.org/2003/xforms">
         <customers>
            <customerData>
               <firstName></firstName>
               <lastName></lastName>
            </customerData>
            <customerData>
               <firstName></firstName>
               <lastName></lastName>
            </customerData>
         <customers>
      <xforms:instance>
   </instances>
   <bindings>
      <bind>
         <ref>[customers][1][firstName]</ref>
         <boundOption>Page1.firstNameField.value</boundOption>
      </bind>
      <bind>
         <ref>[customers][1][lastName]</ref>
         <boundOption>Page1.lastNameField.value</boundOption>
      </bind>
   </bindings>
</xmlmodel>
```

The following button changes the data model so that the form elements are bound to the first data instance rather than the second. When the button is clicked, the *toggle* is triggered, which uses two *set* functions to change the *boundOption* elements in the data model. It also calls the *xmlmodelUpdate* function to ensure that the changes take effect immediately.

```
<button sid="updateDataModel">
   <value>Click to Update Data Model</value>
   <type>select</type>
   <custom:update compute=
      "toggle(activated, 'off', 'on') == '1' &#xA;
      ? set(global.global.xmlmodel[bindings][0][ref], &#xA;
      '[customers][0][firstName]') + &#xA;
      set(global.global.xmlmodel[bindings][1][ref], &#xA;
      '[customers][0][lastName]') + xmlmodelUpdate() &#xA;
      : ''"></custom:update>
</button>
```

9.4 Time and Date Functions

9.4.1 date

Returns a date in “yyyymmdd” format. Either converts a number of seconds (from 12 am, January 1, 1970) or returns the current date if no value is provided.

Syntax

```
date(datesecs)
```

datesecs	*integer*	optional. The number of seconds from 12 am, January 1, 1970. If no value is provided, the current date is returned.

Returns

A string containing the current date, or the date specified by the a date represented by the number of seconds since 00:00:00 GMT, January 1, 1970

Example

In this example, if run on January 18th, 1998, the result of *date* is “19980118”.

```
<field sid="dateTestField">
   <label>Test date()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="date()"></value>
</field>
```

9.4.2 dateToSeconds

Returns the number of seconds from the GMT date and time (represented in date and time respectively) since 00:00:00 GMT, January 1st, 1970.

When passing a *date* parameter to the function *dateToSeconds*, the date should be passed in a format used by XFDL. See the *format* option description for valid date formats.

Note: *The best way to ensure that you pass a date in a valid format is to enter the date in a field, label or cell that has an XFDL format option assigned to it. See the second example below.*

When passing a *time* parameter to the function *dateToSeconds*, the time may consist only of hours:minutes. You may use a 24-hour clock (for example, 23:34) or a 12-hour clock with A.M and P.M. (11:34 P.M.) designators. *Time* is an optional parameter.

An error occurs if either of *date* or *time* is not well formed.

To call the reverse of this function use the *date* function.

Syntax

```
dateToSeconds(date, time)
```

date	*date string*	a date in a recognized format (see section “8.18 format” for a list of formats)
time	*time string*	a time (to minutes) in a recognized format (for example, 23:34 or 11:34 P.M.)

Returns

A string containing the number of seconds, or "" if an error occurs.

Example

In this example, the result of *dateToSeconds* is “890329140”.

```
<field sid="dtsField">
   <value compute="dateToSeconds(’1998-03-19’, ’09:39’)"></value>
</field>
```

The following example shows how to pass in a date that is set in another field.

```
<field sid="enterDateField">
```

```
        <format>
            <ae>date</ae>
            <ae>long</ae>
        </format>
        <value></value>
    </field>
    <field sid="dtsField">
        <value compute="dateToSeconds(enterDateField.value, '09:39')"
            ></value>
    </field>
```

In this example, the first field takes a date as input, and formats it in XFDL's long format. The second field calls the *dateToSeconds* function, and uses an option reference as a parameter (*enterDateField.value*). This reference takes the already-formatted date that the user enters, and passes it into the function.

9.4.3 day

Returns the numeric day of the month for the provided date in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well formed.

Syntax

```
day(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1, 1970. If no value is supplied, the current date is used.

Returns

A string containing the day, or "" if an error occurs.

Example

In this example, the result of *day* is “19”.

```
<field sid="dayTestField">
   <label>Test day()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="day(’890300356’)"></value>
</field>
```

9.4.4 dayOfWeek

Returns the numeric day of the week (Sunday=1, and so on) for the provided date in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well-formed.

Syntax

```
dayOfWeek(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970. If no value is supplied, the current date is used.

Returns

A string containing the day of the week, or "" if an error occurs.

Example

In this example, the result of *dayOfWeek* is “5”.

```
<field sid="dowTestField">
   <label>Test dayOfWeek()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="dayOfWeek(’890300356’)"></value>
</field>
```

9.4.5 endOfMonth

Returns the number of seconds since 00:00:00 GMT, January 1st, 1970 to the current time on the last day of the month in the date provided in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well-formed.

Syntax

```
endOfMonth(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970. If no value is supplied, the current date is used.

Returns

A string containing the number of seconds, or "" if an error occurs.

Example

In this example, the result of *endOfMonth* is “891337156”.

```
<field sid="eomTestField">
   <label>Test endOfMonth()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="endOfMonth(’890300356’)"></value>
</field>
```

9.4.6 hour

Returns the numeric hour for the provided date in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well-formed.

Syntax

```
hour(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970. If no value is supplied, the current date is used.

Returns

A string containing the hour, or "" if an error occurs.

Example

In this example, the result of *hour* is “9”.

```
<field sid="hourTestField">
   <label>Test hour()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="hour(’890300356’)"></value>
</field>
```

9.4.7 minute

Returns the numeric minute for the provided date in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well formed.

Syntax

```
minute(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970. If no value is supplied, the current date is used.

Returns

A string containing the minute, or "" if an error occurs.

Example

In this example, the result of *minute* is “39”.

```
<field sid="minuteTestField">
   <label>Test minute()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="minute(’890300356’)"></value>
</field>
```

9.4.8 month

Returns the numeric month of the year for the provided date in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well formed.

Syntax

```
month(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970. If no value is supplied, the current date is used.

Returns

A string containing the month, or "" if an error occurs.

Example

In this example, the result of *month* is “3”.

```
<field sid="monthTestField">
   <label>Test month()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="month(’890300356’)"></value>
</field>
```

9.4.9 now

Returns the number of seconds since 00:00:00 GMT, January 1st, 1970.

Syntax

```
now()
```

Returns

A string containing the number of seconds.

Example

In this example, if run at 09:39:16 GMT on Thursday, March 19th, 1998, the result of *now* would be “890300356”.

```
<field sid="nowTestField">
   <label>Test now()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="now()"></value>
</field>
```

9.4.10 second

Returns the numeric second for the provided date in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well formed.

Syntax

```
second(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970. If no value is supplied, the current date is used.

Returns

A string containing the second, or "" if an error occurs.

Example

In this example, the result of *second* is “16”.

```
<field sid="secondTestField">
   <label>Test second()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="second(’890300356’)"></value>
</field>
```

9.4.11 time

Returns the current time in “hh:mm AM” format.

Syntax

```
time()
```

Returns

A string containing the current time.

Example

In this example, if run at 3:22 in the afternoon, the result of *time* would be “3:22 PM”.

```
<field sid="timeTestField">
   <label>Test time()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="time()"></value>
</field>
```

9.4.12 year

Returns the numeric year for the provided date in *dateSecs* or the current date if one is not provided. The provided date is a string representing the number of seconds since 00:00:00 GMT, January 1st, 1970.

An error occurs if *dateSecs* is not well-formed.

Syntax

```
year(dateSecs)
```

dateSecs	*number*	optional. A date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970. If no value is supplied, the current date is used.

Returns

A string containing the year, or "" if an error occurs.

Example

In this example, the result of *year* is “1998”.

```
<field sid="yearTestField">
   <label>Test year()</label>
   <format>
      <ae>string</ae>
      <ae>mandatory</ae>
   </format>
   <value compute="year(’890300356’)"></value>
</field>
```

Appendix A: Quick Reference Tables

Table of Items and Form and Page Globals

The following table lists the available options for each item type:

Item	Available Options
action	activated; active; data; datagroup; delay; itemnext; itemprevious; printsettings; transmitdatagroups; transmitformat; transmitgroups; transmititemrefs; transmititems; transmitnamespaces; transmitoptionrefs; transmitoptions; transmitpagerefs; type; url
box	bgcolor; borderwidth; fontinfo; itemlocation; itemnext; itemprevious; printbgcolor; size; visible
button	activated; active; bgcolor; borderwidth; coordinates; data; datagroup; focused; fontcolor; fontinfo; format; help; image; imagemode; itemlocation; itemnext; itemprevious; justify; keypress; mouseover; next; previous; printbgcolor; printfontcolor; printsettings; saveformat; signature; signatureimage; signdatagroups; signer; signformat; signgroups; signitemrefs; signitems; signnamespaces; signoptionrefs; signoptions; signpagerefs; size; transmitdatagroups; transmitformat; transmitgroups; transmititemrefs; transmititems; transmitnamespaces; transmitoptionrefs; transmitoptions; transmitpagerefs; type; url; value; visible
cell	activated; active; data; datagroup; group; itemnext; itemprevious; label; printsettings; saveformat; transmitdatagroups; transmitformat; transmitgroups; transmititemrefs; transmititems; transmitnamespaces; transmitoptionrefs; transmitoptions; transmitpagerefs; type; url; value
check	active; bgcolor; editstate; focused; fontcolor; fontinfo; help; itemlocation; itemnext; itemprevious; keypress; label; labelbgcolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; printbgcolor; printfontcolor; printlabelbgcolor; printlabelfontcolor; size; value, visible
combobox	activated; active; bgcolor; borderwidth; editstate; focused; fontcolor; fontinfo; format; group; help; itemlocation; itemnext; itemprevious; justify; keypress; label; labelbgcolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; printbgcolor; printlabelbgcolor; printfontcolor; printlabelfontcolor; size; value; visible

Item	Available Options
data	datagroup; filename; itemnext; itemprevious; mimedata; mimetype
field	active; bgcolor; borderwidth; editstate; focused; fontcolor; fontinfo; format; help; itemlocation; itemnext; itemprevious; justify; keypress; label; labelbgcolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; printbgcolor; printlabelbgcolor; printfontcolor; printlabelfontcolor; scrollhoriz; scrollvert; size; value; visible
help	active; itemnext; itemprevious; value
label	active; bgcolor; borderwidth; fontcolor; fontinfo; format; help; image; imagemode; itemlocation; itemnext; itemprevious; justify; printbgcolor; printfontcolor; size; value; visible
line	fontcolor; fontinfo; itemlocation; itemnext; itemprevious; printfontcolor; size; thickness; visible
list	active; bgcolor; borderwidth; editstate; focused; fontcolor; fontinfo; format; group; help; itemlocation; itemnext; itemprevious; keypress; label; labelbgcolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; printbgcolor; printlabelbgcolor; printfontcolor; printlabelfontcolor; size; value; visible
popup	activated; active; bgcolor; borderwidth; editstate; focused; fontcolor; fontinfo; format; group; help; itemlocation; itemnext; itemprevious; justify; keypress; label; mouseover; next; previous; printbgcolor; printfontcolor; size; value; visible
radio	active; bgcolor; editstate; focused; fontcolor; fontinfo; group; help; itemlocation; itemnext; itemprevious; keypress; label; labelbgcolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; printbgcolor; printlabelbgcolor; printfontcolor; printlabelfontcolor; size; value; visible
signature	fullname; layoutinfo; itemnext; itemprevious; mimedata; signature; signdatagroups; signer; signformat; signitems; signitemrefs; signgroups; signnamespaces; signoptions; signoptionrefs; signpagerefs
spacer	fontinfo; itemlocation; itemnext; itemprevious; label; size
toolbar	bgcolor; itemnext; itemprevious; mouseover

Item	Available Options
page globals	activated; bgcolor; borderwidth; focused; fontcolor; fontinfo; itemfirst; itemlast; keypress; label; mouseover; next; pagefirst; pageid; pagelast; pagenext; pageprevious; printbgcolor; printsettings; saveformat; transmitformat
form globals	activated; bgcolor; borderwidth; dirtyflag; focused; fontcolor; fontinfo; formid; keypress; next; printbgcolor; printsettings; saveformat; transmitformat; triggeritem; version

Table of Options

The following table list the details for the available options:

Option	Details
acclabel	`<acclabel>`*`message`*`</acclabel>` **Default:** n/a **Items:** button; check; combobox; field; list; popup; radio
activated	`<activated>on|maybe|off</activated>` **Default:** off **Items:** action; button; cell; combobox; popup; page global; form global
active	`<active>on|off</active>` **Default:** on **Items:** action; button; cell; check; combobox; field; help; label; list; popup; radio. **Note:** To prevent user input in a field, use the editstate **readonly**.

Option	Details

bgcolor	`<bgcolor content ="array">`
	`  <ae>`*`color name`*`</ae>`
	`</bgcolor>`
	
	`<bgcolor>`
	`  <ae>`*`R value`*`</ae>`
	`  <ae>`*`G value`*`</ae>`
	`  <ae>`*`B value`*`</ae>`
	`</bgcolor>`
	
	Default:
	• for form — white
	• for page — the form bgcolor setting or default
	• for button — gray
	• for check, field, list, popup, radio — white
	• for label — transparent
	• all other items — the background color of the page
	Items: box; button; check; combobox; field; label; list; popup; radio; toolbar; page globals; form globals

borderwidth	`<borderwidth>`*`width`*`</borderwidth>` **Default:** • for label — 0 • for all other items — the borderwidth set in characteristics, or 1 if no characteristics set **Items:** box; button; combobox; field; label; list; popup; page globals;form globals

coordinates	`<coordinates>`
	`  <ae>`*`X_coordinate`*`</ae>`
	`  <ae>`*`Y_coordinate`*`</ae>`
	`</coordinates>`
	
	Default: none
	Items: button

Option	Details
data	`<data>data_item</data>` **Default:** none **Items:** action; button; cell

datagroup	`<datagroup>`
	`  <ae>datagroup_reference</ae>`
	`  <ae>datagroup reference</ae>`
	`</datagroup>`
	
	Default: none
	
	Items: action; button; cell; data

delay	`<delay>`
	`  <ae>repeat|once<ae>`
	`  <ae>`*`interval`*`</ae>`
	`</delay>`
	
	Default: once with an interval of 0 seconds
	
	Items: action

dirtyflag	`<dirtyflag>on|off</dirtyflag>` **Default:** none **Items:** form globals
editstate	`<editstate>readonly|writeonly|readwrite</editstate>` **Default:** readwrite **Items:** check; combobox; field; list; popup; radio

excludedmetadata	`<excludedmetadata>`
	`  <servernotarizations>`
	`  <ae>`*`Notarization`*`</ae>`
	`  ...`
	`  <ae>`*`Notarization`*`</ae>`
	`  </servernotarizations>`
	`</excludedmetadata>`
	
	Default: none
	
	Items: signature

Option	Details
filename	`<filename>`*`file name`*`</filename>` **Default:** none **Items:** data
focused	`<focused>on|off</focused>` **Default:** off **Items:** button; check; combobox; field; list; popup; radio; page global; form global
focuseditem	`<focuseditem>`*`sid`*`</focuseditem>` **Default:** n/a **Items:** page global

fontcolor	`<fontcolor>`
	`  <ae>`*`color name`*`</ae>`
	`</fontcolor>`
	
	`<fontcolor>`
	`  <ae>`*`R value`*`</ae>`
	`  <ae>`*`G value`*`</ae>`
	`  <ae>`*`B value`*`</ae>`
	`</fontcolor>`
	
	Default:
	• for check and radio — red
	• for all other items, the fontcolor set in the page or form globals, or black if no preference set
	
	Items: button; check; combobox; field; label; line; list; popup; radio; page globals; form globals

Option	Details

fontinfo	`<fontinfo>`
	`  <ae>`*font name*`</ae>`
	`  <ae>`*point size*`</ae>`
	`  <ae>`*weight*`</ae>`
	`  <ae>`*effects*`</ae>`
	`  <ae>`*form*`</ae>`
	`</fontinfo>`
	
	* weight, effects, and form are optional
	
	Default: the fontinfo set in page or form globals, or Helvetica 8 plain if no characteristics set
	
	Items: box; button; check; combobox; field; label; line; list; popup; radio; spacer; page globals; form globals

format	`<format>`
	`  <ae>`*data type*`</ae>`
	`  <ae>`*format flag*`</ae>`
	`  <ae>`*check flag*`</ae>`
	`</format>`
	
	* format and check flags are optional, and multiple flags are valid
	
	Default:
	• for data type — ASCII string
	• for format flag — depends on data type
	• for check option — depends on data type
	
	Items: button; combobox; field; label; list; popup

formid	`<formid>`
	`  <title>`*string*`</title>`
	`  <serialnumber>`*string*`</serialnumber>`
	`  <version>`*AA.Bb.cc*`</version>`
	`</formid>`
	
	* format and check flags are optional, and multiple flags are valid
	
	Default: none
	
	Items: form globals

Option	Details
formid	`<fullname>`*`name`*`</fullname>` **Default:** none **Items:** signature
group	`<group>group name|group reference</group>` **Default:** none **Items:** cell; combobox; list; popup; radio
help	`<help>`*`item reference`*`</help>` **Default:** none **Items:** button; check; combobox; field; label; list; popup; radio
image	`<image>`*`item reference`*`</image>` **Default:** none **Items:** button; label
imagemode	`<imagemode>clip|resize|scale</imagemode>` **Default:** resize **Items:** button; label
itemfirst	`<itemfirst>`*`item reference`*`</itemfirst>` **Default:** none **Items:** page global

Option	Details

itemlocation	`<itemlocation>`
	`  <ae>`
	`  <ae>`*`specification`*`</ae>`
	`  </ae>`
	`  ...`
	`  <ae>`
	`  <ae>`*`specification`*`</ae>`
	`  </ae>`
	`</itemlocation>`
	
	* a specification is usually a modifier and one or more itemtag(s).
	
	Default:
	• for the first item — the top left corner of the form
	• for all other items — vertically below the previously created item and horizontally at the left margin
	
	Items: box; button; check; combobox; field; label; line; list; popup; radio; spacer

itemlast	`<itemlast>`*`item reference`*`</itemlast>` **Default:** none **Items:** page global
itemnext	`<itemnext>`*`item reference`*`</itemnext>` **Default:** none **Items:** action; box; button; cell; check; combobox; data; field; help; label; line; list; popup; radio; signature; spacer; toolbar
itemprevious	`<itemprevious>`*`item reference`*`</itemprevious>` **Default:** none **Items:** action; box; button; cell; check; combobox; data; field; help; label; line; list; popup; radio; signature; spacer; toolbar

Option	Details
justify	`<justify>left|right|center</justify>` **Default:** • for button and popup — center • for label — left **Items:** button; combobox; field; label; popup
keypress	`<keypress>`*`key pressed`*`</keypress>` **Default:** none **Items:** button; check; combobox; field; list; popup; radio; page globals; form globals
label	`<label>`*`label text`*`</label>` **Default:** none **Items:** cell; check; combobox; field; list; popup; radio; spacer; page globals; form globals

labelbgcolor	`<labelbgcolor>`
	`  <ae>`*`color name`*`</ae>`
	`</labelbgcolor>`
	
	`<labelbgcolor>`
	`  <ae>`*`R value`*`</ae>`
	`  <ae>`*`G value`*`</ae>`
	`  <ae>`*`B value`*`</ae>`
	`</labelbgcolor>`
	
	Default: transparent
	
	Items: check; combobox; field; list; radio

labelborderwidth	`<labelborderwidth>`*`width`*`</labelborderwidth>` **Default:** 0 pixels **Items:** check; combobox; field; list; radio

Option	Details

labelfontcolor	`<labelfontcolor>`
	`  <ae>`*color name*`</ae>`
	`</labelfontcolor>`
	
	`<labelfontcolor>`
	`  <ae>`*R value*`</ae>`
	`  <ae>`*G value*`</ae>`
	`  <ae>`*B value*`</ae>`
	`</labelfontcolor>`
	
	Default: the item's labelfontcolor setting, or the global fontcolor setting or default
	
	Items: check; combobox; field; list; radio

labelfontinfo	`<labelfontinfo>`
	`  <ae>`*font name*`</ae>`
	`  <ae>`*point size*`</ae>`
	`  <ae>`*weight*`</ae>`
	`  <ae>`*effects*`</ae>`
	`  <ae>`*form*`</ae>`
	`</labelfontinfo>`
	
	* weight, effects, and form are optional
	
	Default: Helvetica, 8 plain
	
	Items: check; combobox; field; list; radio

layoutinfo	`<layoutinfo>`
	`  <pagehashes>`
	`  <PAGE1>`*pagehash*$_1$`</PAGE1>`
	`  ...`
	`  <PAGEn>`*pagehash*$_n$`</PAGEn>`
	`  </pagehashes>`
	`</layoutinfo>`
	
	Default: none
	
	Items: signature

linespacing	`<linespacing>offset</linespacing>` **Default:** 0 **Items:** button, label, spacer

Option	Details
mimedata	`<mimedata encoding="format">data</mimedata>` **Default:** none **Items:** data, signature
mimetype	`<mimetype>MIME type</mimetype>` **Default:** none **Items:** data
mouseover	`<mouseover>on|off</mouseover>` **Default:** off **Items:** button; check; combobox; field; list; popup; radio; toolbar; page global
next	`<next>item reference</next>` **Default:** • when the form opens — the first non-toolbar item in the form's description that users can modify • when tabbing to subsequent items — the next item in the form's description that users can modify • when tabbing from the last item — the first item in the form's description that users can modify (can be a toolbar item) **Items:** button; combobox; check; field; list; popup; radio; page globals
pagefirst	`<pagefirst>page reference</pagefirst>` **Default:** none **Items:** page global

pageid	`<pageid>`
	`  <serialnumber>string</serialnumber>`
	`</pageid>`
	
	Default: none
	
	Items: form globals

Option	Details
pagelast	`<pagelast>`*`page reference`*`</pagelast>` **Default:** none **Items:** page global
pagenext	`<pagenext>`*`page reference`*`</pagenext>` **Default:** none **Items:** page global
pageprevious	`<pageprevious>`*`page reference`*`</pageprevious>` **Default:** none **Items:** page global
previous	`<previous>`*`item_reference`*`</previous>` **Default:** the previous item in the form description **Items:** button; combobox; check; field; list; popup; radio; page globals

printbgcolor	`<printbgcolor>`
	`  <ae>`*`color name`*`</ae>`
	`</printbgcolor>`
	
	`<printbgcolor>`
	`  <ae>`*`R value`*`</ae>`
	`  <ae>`*`G value`*`</ae>`
	`  <ae>`*`B value`*`</ae>`
	`</printbgcolor>`
	
	Default:
	• for form — white
	• for page — the form setting or default
	• for items — the specified or default item bgcolor
	
	Items: box; button; check; combobox; field; label; list; popup; radio; page global; form global

Option	Details

printfontcolor	`<printfontcolor>`
	`  <ae>`*`color name`*`</ae>`
	`/printfontcolor>`
	
	`<printfontcolor>`
	`  <ae>`*`R value`*`</ae>`
	`  <ae>`*`G value`*`</ae>`
	`  <ae>`*`B value`*`</ae>`
	`</printfontcolor>`
	
	Default:
	• for button, combobox, field, label, line, list, popup: the fontcolor setting or default for the item
	• for radio, check: red
	Items: button; check; combobox; field; label; line; list; popup; radio

printlabelbgcolor	`<printlabelbgcolor>`
	`  <ae>`*`color name`*`</ae>`
	`</printlabelbgcolor>`
	
	`<printlabelbgcolor>`
	`  <ae>`*`R value`*`</ae>`
	`  <ae>`*`G value`*`</ae>`
	`  <ae>`*`B value`*`</ae>`
	`</printlabelbgcolor>`
	
	Default: the item's specified or default labelbgcolor
	Items: check; combobox; field; list; radio

printlabelfontcolor	`<printlabelfontcolor>`
	`  <ae>`*`color name`*`</ae>`
	`</printlabelfontcolor>`
	
	`<printlabelfontcolor>`
	`  <ae>`*`R value`*`</ae>`
	`  <ae>`*`G value`*`</ae>`
	`  <ae>`*`B value`*`</ae>`
	`</printlabelfontcolor>`
	
	Default: the item's specified or default labelfontcolor
	Items: check; combobox; field; list; radio

Option	Details

printsettings	`<printsettings>`
	`  <pages>`*`page list`*`</pages>`
	`  <dailog>`*`dialog settings`*`</dialog>`
	`  <borderwidth>`*`border setting`*`</borderwidth>`
	`</printsettings>`
	
	Default: the page list defaults to include all pages in the form, the dialog defaults to on, has the following settings:
	• orientation — portrait
	• copies — 1
	• printpages active — on
	• printpages choices — all
	• border defaults to off
	
	Items: action, button, cell, page globals, form globals

rtf	`<rtf>`*`rich text string`*`</rtf>` **Default:** a default rich text string is created using the *value*, *fontinfo*, *fontcolor*, *justify*, and *bgcolor* options **Items:** field
saveformat	`<saveformat>`*`mimetype`*`</saveformat>` **Default:** none **Items:** button; cell; form globals; page globals
scrollhoriz	`<scrollhoriz>never|always|wordwrap</scrollhoriz>` **Default:** never **Items:** field
scrollvert	`<scrollvert>never|always|fixed</scrollvert>` **Default:** never **Items:** field

Option	Details
signature	`<signature>`*`string`*`</signature>` **Default:** none **Items:** button; signature
signatureimage	`<signatureimage>`*`data item reference`*`</signatureimage>` **Default:** none **Items:** button

signdatagroups	`<signdatagroups>`
	`  <ae>keep|omit</ae>`
	`  <ae>`*`datagroup reference`*`</ae>`
	`  <ae>`*`datagroup reference`*`</ae>`
	`</signdatagroups>`
	
	Default: keep
	
	Items: button; signature

signdetails	`<signdetails>`
	`  <dialogcolumns>`
	`  <ae>`*`attribute`*$_1$`</ae>`
	`  <ae>`*`attribute`*$_n$`</ae>`
	`  </dialogcolumns>`
	`  <filteridentity>`
	`  <ae>`
	`  <tag>`*`attribute`*$_1$`</tag>`
	`  <value>`*`filter`*$_1$`</value>`
	`  </ae>`
	`  <ae>`
	`  <tag>`*`attribute`*$_n$`</tag>`
	`  <value>`*`filter`*$_n$`</value>`
	`  </ae>`
	`  </filteridentity>`
	`</signdetails>`
	
	Default: all certificates are available, and the owner's common name and email address is shown.
	
	Items: button; signature

Option	Details
signer	`<signer>string</signer>` **Default:** none **Items:** button; signature

signformat	`<signformat>`
	`  MIME type; engine="signature engine"; verifier;`
	`  cval; parameters`
	`</signformat>`
	
	Default: XFDL MIME type; Generic RSA signature engine; Basic verifier; do not sign current values
	
	Items: button; signature

signgroups	`<signgroups>`
	`  <ae>keep|omit</ae>`
	`  <ae>group reference`$_1$`</ae>`
	`  ...`
	`  <ae>group reference`$_n$`</ae>`
	`</signgroups>`
	
	Default: keep
	
	Items: button; signature

signitems	`<signitems>`
	`  <ae>keep|omit</ae>`
	`  <ae>item type`$_1$`</ae>`
	`  ...`
	`  <ae>item type`$_n$`</ae>`
	`</signitems>`
	
	Default: keep
	
	Items: button; signature

Option	Details

signitemrefs	`<signitemrefs>`
	`<ae>keep|omit</ae>`
	`<ae>`*item reference*$_1$`</ae>`
	`...`
	`<ae>`*item reference*$_n$`</ae>`
	`</signitemrefs>`
	
	Default: keep
	
	Items: button; signature

signnamespaces	`<signnamespaces>`
	`<ae>keep|omit</ae>`
	`<ae>`*namespace URI*$_1$`</ae>`
	`...`
	`<ae>`*namespace URI*$_n$`</ae>`
	`</signnamespaces>`
	
	Default: keep
	
	Items: button; signature

signoptionrefs	`<signoptionrefs>`
	`<ae>keep|omit</ae>`
	`<ae>`*option reference*$_1$`</ae>`
	`...`
	`<ae>`*option reference*$_n$`</ae>`
	`</signoptionrefs>`
	
	Default: keep
	
	Items: button; signature

signoptions	`<signoptions>`
	`<ae>keep|omit</ae>`
	`<ae>`*option type*$_1$`</ae>`
	`...`
	`<ae>`*option type*$_n$`</ae>`
	`</signoptions>`
	
	Default: keep
	
	Items: button; signature

Option	Details

signpagerefs	`<signpagerefs>`
	`<ae>keep|omit</ae>`
	`<ae>`*page reference*$_1$`</ae>`
	`...`
	`<ae>`*page reference*$_n$`</ae>`
	`</signpagerefs>`
	
	Default: keep
	
	Items: button; signature

size	`<size>`
	`<ae>`*width*`</ae>`
	`<ae>`*height*`</ae>`
	`</size>`
	
	* the unit of measurement is characters.
	
	Default: see “Appendix B: Default Sizes” on page 379
	
	Items: box; button; check; combobox; field; label; line; list; popup; radio; spacer

texttype	`<texttype> text/plain | text/rtf <texttype>` **Default:** text/plain **Items:** field
thickness	`<thickness>`*thickness*`</thickness>` **Default:** 1 pixel **Items:** line

transmitdatagroups	`<transmitdatagroups>`
	`<ae>keep|omit</ae>`
	`<ae>`*datagroup identifier*$_1$`</ae>`
	`...`
	`<ae>`*datagroup identifier*$_n$`</ae>`
	`</transmitdatagroups>`
	
	Default: keep
	
	Items: action; button; cell

Option	Details
transmitformat	`<transmitformat>`*MIME type*`</transmitformat>` **Default:** application/vnd.xfdl **Items:** action; button; cell; form globals; page globals

transmitgroups	`<transmitgroups>`
	`<ae>keep|omit</ae>`
	`<ae>`*group identifier*$_1$`</ae>`
	`...`
	`<ae>`*group identifier*$_n$`</ae>`
	`</transmitgroups>`
	
	Default: keep
	
	Items: action; button; cell

transmititemrefs	`<transmititemrefs>`
	`<ae>keep|omit</ae>`
	`<ae>`*item identifier*$_1$`</ae>`
	`...`
	`<ae>`*item identifier*$_n$`</ae>`
	`</transmititemrefs>`
	
	Default: keep
	
	Items: action; button; cell

transmititems	`<transmititems>`
	`<ae>keep|omit</ae>`
	`<ae>`*item type*$_1$`</ae>`
	`...`
	`<ae>`*item type*$_n$`</ae>`
	`</transmititems>`
	
	Default: keep
	
	Items: action; button; cell

Option	Details

transmitnamespaces	`<transmitnamespaces>`
	`<ae>keep|omit</ae>`
	`<ae>`*namespace URI*$_1$`</ae>`
	`...`
	`<ae>`*namespace URI*$_n$`</ae>`
	`</transmitnamespaces>`
	
	Default: keep
	
	Items: action; button; cell

transmitoptionrefs	`<transmitoptionrefs>`
	`<ae>keep|omit</ae>`
	`<ae>`*option identifier*$_1$`</ae>`
	`...`
	`<ae>`*option identifier*$_n$`</ae>`
	`</transmitoptionrefs>`
	
	Default: keep
	
	Items: action; button; cell

transmitoptions	`<transmitoptions>`
	`<ae>keep|omit</ae>`
	`<ae>`*option type*$_1$`</ae>`
	`...`
	`<ae>`*option type*$_n$`</ae>`
	`</transmitoptions>`
	
	Default: keep
	
	Items: action; button; cell

transmitpagerefs	`<transmitpagerefs>`
	`<ae>keep|omit</ae>`
	`<ae>`*page identifier*$_1$`</ae>`
	`...`
	`<ae>`*page identifier*$_n$`</ae>`
	`</transmitpagerefs>`
	
	Default: keep
	
	Items: action; button; cell

Option	Details
triggeritem	`<triggeritem>`*`item reference`*`</triggeritem>` **Default:** the item reference of the item that triggered the submit or done **Items:** form globals
type	`<type>`*`action type`*`</type>` **Default:** select **Items:** action; button; cell

url	`<url>`
	`  <ae>`*`URL|item reference`*`</ae>`
	`</url>`
	
	Default: none
	Items: action; button; cell

value	`<value>`*`setting`*`</value>` **Default:** depends on item **Items:** button; cell; check; combobox; field; help; label; list; popup; radio
visible	`<visible>on|off</visible>` **Default:** on **Items:** box; button; check; combobox; field; label; line; list; popup; radio

Cross Reference Table for Items and Options

	action	box	button	cell	check	combobox	data	field	help	label	line	list	popup	radio	signature	spacer	toolbar	page global	form global
acclabel			•		•	•		•				•	•	•					
activated	•		•	•		•							•					•	•
active	•		•	•	•	•		•	•	•		•	•	•					
bgcolor		•	•		•	•		•		•		•	•	•			•	•	•
borderwidth		•	•			•		•		•		•	•					•	•
coordinates			•																
data	•		•	•															
datagroup	•		•	•			•												
delay	•																		
dirtyflag																			•
editstate					•	•		•				•	•	•					
excludedmetadata															•				
filename							•												
focused			•		•	•		•				•	•	•				•	•
focuseditem																		•	
fontcolor			•		•	•		•		•	•	•	•	•				•	•
fontinfo		•	•		•	•		•		•	•	•	•	•		•		•	•
format			•			•		•		•		•	•						
formid																			•
fullname															•				
group				•		•						•	•	•					
help			•		•	•		•		•		•	•	•					
image			•							•									
imagemode			•							•									
itemfirst																		•	
itemlast																		•	
itemlocation		•	•		•	•		•		•	•	•	•	•		•			
itemnext	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		
itemprevious	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		
justify			•			•		•		•			•						
keypress			•		•	•		•				•	•	•				•	•
label				•	•	•		•				•	•	•		•		•	•
lebelbgcolor					•	•		•				•		•				•	•
labelborderwidth					•	•		•				•		•					

	action	box	button	cell	check	combobox	data	field	help	label	line	list	popup	radio	signature	spacer	toolbar	page global	form global
labelfontcolor					•	•		•				•		•				•	•
labelfontinfo					•	•		•				•		•				•	•
layoutinfo															•				
linespacing			•							•						•			
mimedata							•								•				
mimetype							•												
mouseover			•		•	•		•				•	•	•			•	•	
next			•		•	•		•				•	•	•				•	
pagefirst																		•	
pageid																		•	
pagelast																		•	
pagenext																		•	
pageprevious																		•	
previous			•		•	•		•				•	•	•				•	
printbgcolor		•	•		•	•		•		•		•	•	•				•	•
printfontcolor			•		•	•		•		•	•	•	•	•					
printlabelbgcolor					•	•		•				•		•					
printlabelfontcolor					•	•		•				•		•					
printsettings	•		•	•														•	•
rtf								•											
saveformat			•	•														•	•
scrollhoriz								•											
scrollvert								•											
signature			•												•				
signatureimage			•																
signdatagroups			•												•				
signdetails			•												•				
signer			•												•				
signformat			•												•				
signgroups			•												•				
signitemrefs			•												•				
signitems			•												•				
signnamespaces			•												•				
signoptionrefs			•												•				
signoptions			•												•				
signpagerefs			•												•				
size		•	•		•	•		•		•	•	•	•	•		•			

	action	box	button	cell	check	combobox	data	field	help	label	line	list	popup	radio	signature	spacer	toolbar	page global	form global
texttype								•											
thickness											•								
transmitdatagroups	•		•	•															
transmitformat	•		•	•														•	•
transmitgroups	•		•	•															
transmititemrefs	•		•	•															
transmititems	•		•	•															
transmitnamespaces	•		•	•															
transmitoptionrefs	•		•	•															
transmitoptions	•		•	•															
transmitpagerefs	•		•	•															
triggeritem																			•
type	•		•	•															
url	•		•	•															
value			•	•	•	•		•	•	•		•	•	•					
visible		•	•		•	•		•		•	•	•	•	•					

Appendix B: Default Sizes

The following table shows the default basic item and bounding box sizes:

Item	Default Item Size	Bounding Box Size
box	**width:** 1 character **height:** 1 character Smaller than 1 not allowed in either dimension	Same as default item size
button	**width:** width of text **height:** height of text (text is the *value* option) **or** size of embedded image if it exists	Same as default item size
check box	**width:** 1 character **height:** 1 character	**width:** larger of 1 character and label width **height:** label height plus 1 character
combobox	**width:** larger of label width and widest cell [3] **height:** 1 character	Same as default item size [2]
field	**width:** 30 characters **height:** 1 character	**width:** larger of item width and label width [2] **height:** height of item plus height of label [2]
label	**width:** 1 character if label empty, otherwise label width **height:** 1 character if label empty, otherwise label height **or** size of embedded image if it exists	Same as default item size

Item	Default Item Size	Bounding Box Size
line	**width:** 30 character **height:** 1 pixel One dimension must be 0 [1]	Same as default item size
list	**width:** larger of label width and widest cell [3] **height:** number of cells in list	**width:** larger of item widthand widest cell [2] **height:** height of item plusheight of label
popup	**width:** larger of label width and widest cell [3] **height:** 1 character	Same as default item size [2]
radio button	**width:** 1 character **height:** 1 character	**width:** larger of 1 character and label width **height:** label height plus 1 character
spacer	**width:** 1 character if label empty, otherwise label width **height:** 1 character if label empty, otherwise label height (label is invisible)	Same as default item size

Usage Notes

1. For *line* items, either height or width must be set to zero. The *thickness* option specifies the thickness (in pixels) of the line in the dimension containing zero (0).
2. This includes a scroll bar if one appears.
3. The cell's width comes from the cell's *value* option setting.

Appendix C: Order of Precedence of Filters

Both signature and transmission filters may conflict with each other. For example, you might create a signature button that omits all fields in the signitems option, but also has a signitemrefs option that stipulates *FIELD1* should be signed. These filters are applied in the following order:

Filter	Behavior If keep flag is used	Behavior If omit flag is used	Notes
1. Filter namespaces, based on *transmitnamespaces* or *signnamespaces* setting	Keeps only elements and attributes in the namespaces indicated; throws others out.	Omits only elements and attributes in the namespaces indicated; throws them out.	An element is kept if any of its children are kept, even if it is in the wrong namespace.
2. Filter types of items, based on *transmititems* or *signitems* setting	Keeps only those types indicated; throws others out, including their options.	Omits only those types incidated; throws them out, including their options.	
3. Filter types of options based on *transmitoptions* and *signoptions* setting	In the items that remain, keeps all option types indicated; throws others out.	In the items that remain, omits all option types indicated.	
4. Filter specific pages based on *transmitpagerefs* or *signpagerefs* settings	Keeps the pages whose tags are specified. Settings in *transmitnamespaces*, *transmititems*, and *transmitoptions* are respected.	Omits the pages whose tags are specified. Overrides settings in *transmitnamespaces*, *transmititems*, and *transmitoptions.*	The page does not entirely disappear from the source code; the page tags still exist

Filter	Behavior If keep flag is used	Behavior If omit flag is used	Notes
5. Filter groups of items based on *transmitdatagroups* and *transmitgroups*, or *signdatagroups* and *signgroups* settings	Keeps those items whose tags are specified, even if the items are of a type that should not be kept according to a *transmitnamespaces* or *transmititems* setting.	Omits those items whose tags are specified, even if the items are of a type that should be kept according to a *transmitnamespaces* or *transmititems* setting.	This option's settings override those in *transmitpagerefs* and *signpagerefs*.
6. Filter specific items based on *transmititemrefs* or *signitemrefs* settings	Keeps the items whose tags are specified; overrides previous settings if necessary. Settings in *transmitoptions* and *signoptions* are respected.	Omits the items specified; overrides the previous settings if necessary.	This option's settings override those in *transmitnamespaces*, *transmititems*, *transmitgroups*, *transmitpagerefs* and *transmitdatagroups* or *signitems*, *signgroups*, *signpagerefs* and *signdatagroups*.
7. Filter specific options based on *transmitoptionrefs* and *signoptionrefs* setting	Regardless of all other settings above, keeps the specific option instances referred to; does not keep any other options; in the case of items that will be omitted except for a single option, the item will be kept, with its original sid, and only the one option.	Omits the options specified; overrides the previous settings if necessary.	This option's settings override all other filters, including *transmitnamespaces*, *transmititems*, *transmitdatagroups*, *transmitgroups*, *transmititemrefs*, *transmitpagerefs*, *transmitoptions* or *signitems*, *signdatagroups*, *signgroups*, *signitemrefs*, *signpagerefs*, and *signoptions*.

Example

This example uses the transmit-family of options. The order of precedence would be the same for the sign-family of options.

```
<page sid="Page1">
   <global sid="global"></global>
   <button sid="submitButton">
      <value>Filter Submission</value>
      <type>done</type>
      <url>
         <ae>http://www.server.dmn/cgi-bin/processForm</ae>
      </url>
      <transmitnamespaces>
         <ae>omit</ae>
         <ae>http://www.PureEdge.com/XFDL/Custom</ae>
      </transmitnamespaces>
      <transmititems>
         <ae>omit</ae>
         <ae>data</ae>
      </transmititems>
      <transmitdatagroups>
         <ae>keep</ae>
         <ae>enclosures</ae>
         <ae>related</ae>
      </transmitdatagroups>
      <transmititemrefs>
         <ae>omit</ae>
         <ae>Page1.data2</ae>
      </transmititemrefs>
      <transmitoptions>
         <ae>omit</ae>
         <ae>filename</ae>
      </transmitoptions>
   </button>
   <button sid="encloseButton">
      <image>encloseImageData</image>
      <type>enclose</type>
      <datagroup>
         <ae>enclosures</ae>
         <ae>related</ae>
      </datagroup>
   </button>
   <data sid="data1">
      <custom:id>324</custom:id>
      <datagroup>
         <ae>enclosures</ae>
      </datagroup>
      <filename>jobdescr.frm</filename>
```

```
        <mimedata encoding="base64-gzip">dfksdfsdfhsdhs</mimedata>
    </data>
    <data sid="data2">
        <datagroup>
            <ae>related</ae>
        </datagroup>
        <filename>resume.doc</filename>
        <mimedata encoding="base64-gzip">dfhsjdfsjhfjs</mimedata>
    </data>
    <data sid="encloseImageData">
        <filename>c:\images\enclose.jpg</filename>
        <mimedata encoding="base64-gzip">
            aswWWW8MjfbyhsUELKKEFir8dfdUUUmnskshie3mkjkkeiIIUIUOl
            fRlgdsoepgejgjj1sd/3/6nnII/fjkess9Wfgjgkggkllgakkk2kl
        </mimedata>
    </data>
</page>
```

As a result of the filtering, the following would happen (see result form description below):

- The custom “id” option would be stripped from the “data1” item, as a result of the *transmitnamespaces* setting.
- The “encloseImageData” data item would be stripped from the form, as a result of the *transmititems* setting.
- The “data1” data item would remain in the form, as a result of the *transmitdatagroups* setting, but would not contain the custom “id” option.
- The “data2” data item would be stripped from the form, as a result of the *transmititemrefs* setting.
- The *filename* option would be stripped from “data1”, as a result of the *transmitoptions* setting.

The form description that would be received once filtering was applied would look like this:

```
<page sid="page1">
    <global sid="global"></global>
    <button sid="submitButton">
        <value>Filter Submission</value>
        <type>done</type>
        <url>
            <ae>http://www.server.dmn/cgi-bin/processForm</ae>
        </url>
        <transmitnamespaces>
            <ae>omit</ae>
            <ae>http://www.PureEdge.com/XFDL/Custom</ae>
        </transmitnamespaces>
        <transmititems>
            <ae>omit</ae>
            <ae>data</ae>
        </transmititems>
        <transmitdatagroups>
```

```
            <ae>keep</ae>
            <ae>enclosures</ae>
            <ae>related</ae>
        </transmitdatagroups>
        <transmititemrefs>
            <ae>omit</ae>
            <ae>page1.data2</ae>
        </transmititemrefs>
        <transmitoptions>
            <ae>omitfilename</ae>
        </transmitoptions>
    </button>
    <button sid="encloseButton">
        <image>encloseImageData</image>
        <type>enclose</type>
        <datagroup>
            <ae>enclosures</ae>
            <ae>related</ae>
        </datagroup>
    </button>
    <data sid="data1">
        <datagroup>
            <ae>enclosures</ae>
        </datagroup>
        <mimedata encoding="base64-gzip">dfksdfsdfhsdhs</mimedata>
    </data>
</page>
```

Appendix D: Color Table

You can specify a color using either the color's name or its Red, Green, and Blue (RGB) triplet. Each value in the RGB triplet is a number from 0 to 255 inclusive, representing the amount of primary color (red, green or blue) required to produce the secondary color. Zero represents the least amount of a color and 255 represents the greatest amount of a color.

For example, the statement:

```
<bgcolor>
   <ae>255</ae>
   <ae>255</ae>
   <ae>255</ae>
</bgcolor>
```

is equivalent to:

```
<bgcolor>
   <ae>white</ae>
</bgcolor>
```

Note: *The transparent color has no RGB equivalent.*

The following pages list the names and RGB triplet values for the available colors.

RGB	Color Name
240 248 255	alice blue
240 248 255	AliceBlue
250 235 215	antique white
250 235 215	AntiqueWhite
255 239 219	AntiqueWhite1
238 223 204	AntiqueWhite2
205 192 176	AntiqueWhite3
139 131 120	AntiqueWhite4
127 255 212	aquamarine
127 255 212	aquamarine1
118 238 198	aquamarine2
102 205 170	aquamarine3
69 139 116	aquamarine4
240 255 255	azure
240 255 255	azure1
224 238 238	azure2
193 205 205	azure3
131 139 139	azure4
245 245 220	beige
255 228 196	bisque
255 228 196	bisque1
238 213 183	bisque2
205 183 158	bisque3
139 125 107	bisque4
0 0 0	black
255 235 205	blanched almond
255 235 205	BlanchedAlmond
0 0 255	blue

RGB	Color Name
138 43 226	blue violet
0 0 255	blue1
0 0 238	blue2
0 0 205	blue3
0 0 139	blue4
138 43 226	BlueViolet
165 42 42	brown
255 64 64	brown1
238 59 59	brown2
205 51 51	brown3
139 35 35	brown4
222 184 135	burlywood
255 211 155	burlywood1
238 197 145	burlywood2
205 170 125	burlywood3
139 115 85	burlywood4
95 158 160	cadet blue
95 158 160	CadetBlue
152 245 255	CadetBlue1
142 229 238	CadetBlue2
122 197 205	CadetBlue3
83 134 139	CadetBlue4
127 255 0	chartreuse
127 255 0	chartreuse1
118 238 0	chartreuse2
102 205 0	chartreuse3
69 139 0	chartreuse4
210 105 30	chocolate

RGB	Color Name
255 127 36	chocolate1
238 118 33	chocolate2
205 102 29	chocolate3
139 69 19	chocolate4
255 127 80	coral
255 114 86	coral1
238 106 80	coral2
205 91 69	coral3
139 62 47	coral4
100 149 237	cornflower blue
100 149 237	CornflowerBlue
255 248 220	cornsilk
255 248 220	cornsilk1
238 232 205	cornsilk2
205 200 177	cornsilk3
139 136 120	cornsilk4
0 255 255	cyan
0 255 255	cyan1
0 238 238	cyan2
0 205 205	cyan3
0 139 139	cyan4
184 134 11	dark goldenrod
0 100 0	dark green
189 183 107	dark khaki
85 107 47	dark olive green
255 140 0	dark orange
153 50 204	dark orchid
233 150 122	dark salmon

RGB	Color Name
143 188 143	dark sea green
72 61 139	dark slate blue
47 79 79	dark slate gray
47 79 79	dark slate grey
0 206 209	dark turquoise
148 0 211	dark violet
184 134 11	DarkGoldenrod
255 185 15	DarkGoldenrod1
238 173 14	DarkGoldenrod2
205 149 12	DarkGoldenrod3
139 101 8	DarkGoldenrod4
0 100 0	DarkGreen
189 183 107	DarkKhaki
85 107 47	DarkOliveGreen
202 255 112	DarkOliveGreen1
188 238 104	DarkOliveGreen2
162 205 90	DarkOliveGreen3
110 139 61	DarkOliveGreen4
255 140 0	DarkOrange
255 127 0	DarkOrange1
238 118 0	DarkOrange2
205 102 0	DarkOrange3
139 69 0	DarkOrange4
153 50 204	DarkOrchid
191 62 255	DarkOrchid1
178 58 238	DarkOrchid2
154 50 205	DarkOrchid3
104 34 139	DarkOrchid4

RGB	Color Name
233 150 122	DarkSalmon
143 188 143	DarkSeaGreen
193 255 193	DarkSeaGreen1
180 238 180	DarkSeaGreen2
155 205 155	DarkSeaGreen3
105 139 105	DarkSeaGreen4
72 61 139	DarkSlateBlue
47 79 79	DarkSlateGray
151 255 255	DarkSlateGray1
141 238 238	DarkSlateGray2
121 205 205	DarkSlateGray3
82 139 139	DarkSlateGray4
47 79 79	DarkSlateGrey
0 206 209	DarkTurquoise
148 0 211	DarkViolet
255 20 147	deep pink
0 191 255	deep sky blue
255 20 147	DeepPink
255 20 147	DeepPink1
238 18 137	DeepPink2
205 16 118	DeepPink3
139 10 80	DeepPink4
0 191 255	DeepSkyBlue
0 191 255	DeepSkyBlue1
0 178 238	DeepSkyBlue2
0 154 205	DeepSkyBlue3
0 104 139	DeepSkyBlue4
105 105 105	dim gray

RGB	Color Name
105 105 105	dim grey
105 105 105	DimGray
105 105 105	DimGrey
30 144 255	dodger blue
30 144 255	DodgerBlue
30 144 255	DodgerBlue1
28 134 238	DodgerBlue2
24 116 205	DodgerBlue3
16 78 139	DodgerBlue4
178 34 34	firebrick
255 48 48	firebrick1
238 44 44	firebrick2
205 38 38	firebrick3
139 26 26	firebrick4
255 250 240	floral white
255 250 240	FloralWhite
34 139 34	forest green
34 139 34	ForestGreen
220 220 220	gainsboro
248 248 255	ghost white
248 248 255	GhostWhite
255 215 0	gold
255 215 0	gold1
238 201 0	gold2
205 173 0	gold3
139 117 0	gold4
218 165 32	goldenrod
255 193 37	goldenrod1

RGB	Color Name
238 180 34	goldenrod2
205 155 29	goldenrod3
139 105 20	goldenrod4
192 192 192	gray
0 0 0	gray0
3 3 3	gray1
26 26 6	gray10
255 255 255	gray100
28 28 28	gray11
31 31 31	gray12
33 33 33	gray13
36 36 36	gray14
38 38 38	gray15
41 41 41	gray16
43 43 43	gray17
46 46 46	gray18
48 48 48	gray19
5 5 5	gray2
51 51 51	gray20
54 54 54	gray21
56 56 56	gray22
59 59 59	gray23
61 61 61	gray24
64 64 64	gray25
66 66 66	gray26
69 69 69	gray27
71 71 71	gray28
74 74 74	gray29

RGB	Color Name
8 8 8	gray3
77 77 77	gray30
79 79 79	gray31
82 82 82	gray32
84 84 84	gray33
87 87 87	gray34
89 89 89	gray35
92 92 92	gray36
94 94 94	gray37
97 97 97	gray38
99 99 99	gray39
10 10 10	gray4
102 102 102	gray40
105 105 105	gray41
107 107 107	gray42
110 110 110	gray43
112 112 112	gray44
115 115 115	gray45
117 117 117	gray46
120 120 120	gray47
122 122 122	gray48
125 125 125	gray49
13 13 13	gray5
127 127 127	gray50
130 130 130	gray51
133 133 133	gray52
135 135 135	gray53
138 138 138	gray54

RGB	Color Name
140 140 140	gray55
143 143 143	gray56
145 145 145	gray57
148 148 148	gray58
150 150 150	gray59
15 15 15	gray6
153 153 153	gray60
156 156 156	gray61
158 158 158	gray62
161 161 161	gray63
163 163 163	gray64
166 166 166	gray65
168 168 168	gray66
171 171 171	gray67
173 173 173	gray68
176 176 176	gray69
18 18 18	gray7
179 179 179	gray70
181 181 181	gray71
184 184 184	gray72
186 186 186	gray73
189 189 189	gray74
191 191 191	gray75
194 194 194	gray76
196 196 196	gray77
199 199 199	gray78
201 201 201	gray79
20 20 20	gray8

RGB	Color Name
204 204 204	gray80
207 207 207	gray81
209 209 209	gray82
212 212 212	gray83
214 214 214	gray84
217 217 217	gray85
219 219 219	gray86
222 222 222	gray87
224 224 224	gray88
227 227 227	gray89
23 23 23	gray9
229 229 229	gray90
232 232 232	gray91
235 235 235	gray92
237 237 237	gray93
240 240 240	gray94
242 242 242	gray95
245 245 245	gray96
247 247 247	gray97
250 250 250	gray98
252 252 252	gray99
0 255 0	green
173 255 47	green yellow
0 255 0	green1
0 238 0	green2
0 205 0	green3
0 139 0	green4
173 255 47	GreenYellow

RGB	Color Name
192 192 192	grey
0 0 0	grey0
3 3 3	grey1
26 26 26	grey10
255 255 255	grey100
28 28 28	grey11
31 31 31	grey12
33 33 33	grey13
36 36 36	grey14
38 38 38	grey15
41 41 41	grey16
43 43 43	grey17
46 46 46	grey18
48 48 48	grey19
5 5 5	grey2
51 51 51	grey20
54 54 54	grey21
56 56 56	grey22
59 59 59	grey23
61 61 61	grey24
64 64 64	grey25
66 66 66	grey26
69 69 69	grey27
71 71 71	grey28
74 74 74	grey29
8 8 8	grey3
77 77 77	grey30
79 79 79	grey31

RGB	Color Name
82 82 82	grey32
84 84 84	grey33
87 87 87	grey34
89 89 89	grey35
92 92 92	grey36
94 94 94	grey37
97 97 97	grey38
99 99 99	grey39
10 10 10	grey4
102 102 102	grey40
105 105 105	grey41
107 107 107	grey42
110 110 110	grey43
112 112 112	grey44
115 115 115	grey45
117 117 117	grey46
120 120 120	grey47
122 122 122	grey48
125 125 125	grey49
13 13 13	grey5
127 127 127	grey50
130 130 130	grey51
133 133 133	grey52
135 135 135	grey53
138 138 138	grey54
140 140 140	grey55
143 143 143	grey56
145 145 145	grey57

RGB	Color Name
148 148 148	grey58
150 150 150	grey59
15 15 15	grey6
153 153 153	grey60
156 156 156	grey61
158 158 158	grey62
161 161 161	grey63
163 163 163	grey64
166 166 166	grey65
168 168 168	grey66
171 171 171	grey67
173 173 173	grey68
176 176 176	grey69
18 18 18	grey7
179 179 179	grey70
181 181 181	grey71
184 184 184	grey72
186 186 186	grey73
189 189 189	grey74
191 191 191	grey75
194 194 194	grey76
196 196 196	grey77
199 199 199	grey78
201 201 201	grey79
20 20 20	grey8
204 204 204	grey80
207 207 207	grey81
209 209 209	grey82

RGB	Color Name
212 212 212	grey83
214 214 214	grey84
217 217 217	grey85
219 219 219	grey86
222 222 222	grey87
224 224 224	grey88
227 227 227	grey89
23 23 23	grey9
229 229 229	grey90
232 232 232	grey91
235 235 235	grey92
237 237 237	grey93
240 240 240	grey94
242 242 242	grey95
245 245 245	grey96
247 247 247	grey97
250 250 250	grey98
252 252 252	grey99
240 255 240	honeydew
240 255 240	honeydew1
224 238 224	honeydew2
193 205 193	honeydew3
131 139 131	honeydew4
255 105 180	hot pink
255 105 180	HotPink
255 110 180	HotPink1
238 106 167	HotPink2
205 96 144	HotPink3

RGB	Color Name
139 58 98	HotPink4
205 92 92	indian red
205 92 92	IndianRed
255 106 106	IndianRed1
238 99 99	IndianRed2
205 85 85	IndianRed3
139 58 58	IndianRed4
255 255 240	ivory
255 255 240	ivory1
238 238 224	ivory2
205 205 193	ivory3
139 139 131	ivory4
240 230 140	khaki
255 246 143	khaki1
238 230 133	khaki2
205 198 115	khaki3
139 134 78	khaki4
230 230 250	lavender
255 240 245	lavender blush
255 240 245	LavenderBlush
255 240 245	LavenderBlush1
238 224 229	LavenderBlush2
205 193 197	LavenderBlush3
139 131 134	LavenderBlush4
124 252 0	lawn green
124 252 0	LawnGreen
255 250 205	lemon chiffon
255 250 205	LemonChiffon

RGB	Color Name
255 250 205	LemonChiffon1
238 233 191	LemonChiffon2
205 201 165	LemonChiffon3
139 137 112	LemonChiffon4
173 216 230	light blue
240 128 128	light coral
224 255 255	light cyan
238 221 130	light goldenrod
250 250 210	light goldenrod yellow
211 211 211	light gray
211 211 211	light grey
255 182 193	light pink
255 160 122	light salmon
32 178 170	light sea green
135 206 250	light sky blue
132 112 255	light slate blue
119 136 153	light slate gray
119 136 153	light slate grey
176 196 222	light steel blue
255 255 224	light yellow
173 216 230	LightBlue
191 239 255	LightBlue1
178 223 238	LightBlue2
154 192 205	LightBlue3
104 131 139	LightBlue4
240 128 128	LightCoral
224 255 255	LightCyan
224 255 255	LightCyan1

RGB	Color Name
209 238 238	LightCyan2
180 205 205	LightCyan3
122 139 139	LightCyan4
238 221 130	LightGoldenrod
255 236 139	LightGoldenrod1
238 220 130	LightGoldenrod2
205 190 112	LightGoldenrod3
139 129 76	LightGoldenrod4
250 250 210	LightGoldenrod Yellow
211 211 211	LightGray
211 211 211	LightGrey
255 182 193	LightPink
255 174 185	LightPink1
238 162 173	LightPink2
205 140 149	LightPink3
139 95 101	LightPink4
255 160 122	LightSalmon
255 160 122	LightSalmon1
238 149 114	LightSalmon2
205 129 98	LightSalmon3
139 87 66	LightSalmon4
32 178 170	LightSeaGreen
135 206 250	LightSkyBlue
176 226 255	LightSkyBlue1
164 211 238	LightSkyBlue2
141 182 205	LightSkyBlue3
96 123 139	LightSkyBlue4
132 112 255	LightSlateBlue

RGB	Color Name
119 136 153	LightSlateGray
119 136 153	LightSlateGrey
176 196 222	LightSteelBlue
202 225 255	LightSteelBlue1
188 210 238	LightSteelBlue2
162 181 205	LightSteelBlue3
110 123 139	LightSteelBlue4
255 255 224	LightYellow
255 255 224	LightYellow1
238 238 209	LightYellow2
205 205 180	LightYellow3
139 139 122	LightYellow4
50 205 50	lime green
50 205 50	LimeGreen
250 240 230	linen
255 0 255	magenta
255 0 255	magenta1
238 0 238	magenta2
205 0 205	magenta3
139 0 139	magenta4
176 48 96	maroon
255 52 179	maroon1
238 48 167	maroon2
205 41 144	maroon3
139 28 98	maroon4
102 205 170	medium aquamarine
0 0 205	medium blue
186 85 211	medium orchid

RGB	Color Name
147 112 219	medium purple
60 179 113	medium sea green
123 104 238	medium slate blue
0 250 154	medium spring green
72 209 204	medium turquoise
199 21 133	medium violet red
102 205 170	Medium Aquamarine
0 0 205	MediumBlue
186 85 211	MediumOrchid
224 102 255	MediumOrchid1
209 95 238	MediumOrchid2
180 82 205	MediumOrchid3
122 55 139	MediumOrchid4
147 112 219	MediumPurple
171 130 255	MediumPurple1
159 121 238	MediumPurple2
137 104 205	MediumPurple3
93 71 139	MediumPurple4
60 179 113	MediumSeaGreen
123 104 238	MediumSlateBlue
0 250 154	MediumSpring Green
72 209 204	MediumTurquoise
199 21 133	MediumVioletRed
25 25 112	midnight blue
25 25 112	MidnightBlue
245 255 250	mint cream
245 255 250	MintCream

RGB	Color Name
255 228 225	misty rose
255 228 225	MistyRose
255 228 225	MistyRose1
238 213 210	MistyRose2
205 183 181	MistyRose3
139 125 123	MistyRose4
255 228 181	moccasin
255 222 173	navajo white
255 222 173	NavajoWhite
255 222 173	NavajoWhite1
238 207 161	NavajoWhite2
205 179 139	NavajoWhite3
139 121 94	NavajoWhite4
0 0 128	navy
0 0 128	navy blue
0 0 128	NavyBlue
253 245 230	old lace
253 245 230	OldLace
107 142 35	olive drab
107 142 35	OliveDrab
192 255 62	OliveDrab1
179 238 58	OliveDrab2
154 205 50	OliveDrab3
105 139 34	OliveDrab4
255 165 0	orange
255 69 0	orange red
255 165 0	orange1
238 154 0	orange2

RGB	Color Name
205 133 0	orange3
139 90 0	orange4
255 69 0	OrangeRed
255 69 0	OrangeRed1
238 64 0	OrangeRed2
205 55 0	OrangeRed3
139 37 0	OrangeRed4
218 112 214	orchid
255 131 250	orchid1
238 122 233	orchid2
205 105 201	orchid3
139 71 137	orchid4
238 232 170	pale goldenrod
152 251 152	pale green
175 238 238	pale turquoise
219 112 147	pale violet red
238 232 170	PaleGoldenrod
152 251 152	PaleGreen
154 255 154	PaleGreen1
144 238 144	PaleGreen2
124 205 124	PaleGreen3
84 139 84	PaleGreen4
175 238 238	PaleTurquoise
187 255 255	PaleTurquoise1
174 238 238	PaleTurquoise2
150 205 205	PaleTurquoise3
102 139 139	PaleTurquoise4
219 112 147	PaleVioletRed

RGB	Color Name
255 130 171	PaleVioletRed1
238 121 159	PaleVioletRed2
205 104 137	PaleVioletRed3
139 71 93	PaleVioletRed4
255 239 213	papaya whip
255 239 213	PapayaWhip
255 218 185	peach puff
255 218 185	PeachPuff
255 218 185	PeachPuff1
238 203 173	PeachPuff2
205 175 149	PeachPuff3
139 119 101	PeachPuff4
205 133 63	peru
255 192 203	pink
255 181 197	pink1
238 169 184	pink2
205 145 158	pink3
139 99 108	pink4
221 160 221	plum
255 187 255	plum1
238 174 238	plum2
205 150 205	plum3
139 102 139	plum4
176 224 230	powder blue
176 224 230	PowderBlue
160 32 240	purple
155 48 255	purple1
145 44 238	purple2

RGB	Color Name
125 38 205	purple3
85 26 139	purple4
255 0 0	red
255 0 0	red1
238 0 0	red2
205 0 0	red3
139 0 0	red4
188 143 143	rosy brown
188 143 143	RosyBrown
255 193 193	RosyBrown1
238 180 180	RosyBrown2
205 155 155	RosyBrown3
139 105 105	RosyBrown4
65 105 225	royal blue
65 105 225	RoyalBlue
72 118 255	RoyalBlue1
67 110 238	RoyalBlue2
58 95 205	RoyalBlue3
39 64 139	RoyalBlue4
139 69 19	saddle brown
139 69 19	SaddleBrown
250 128 114	salmon
255 140 105	salmon1
238 130 98	salmon2
205 112 84	salmon3
139 76 57	salmon4
244 164 96	sandy brown
244 164 96	SandyBrown

RGB	Color Name
46 139 87	sea green
46 139 87	SeaGreen
84 255 159	SeaGreen1
78 238 148	SeaGreen2
67 205 128	SeaGreen3
46 139 87	SeaGreen4
255 245 238	seashell
255 245 238	seashell1
238 229 222	seashell2
205 197 191	seashell3
139 134 130	seashell4
160 82 45	sienna
255 130 71	sienna1
238 121 66	sienna2
205 104 57	sienna3
139 71 38	sienna4
135 206 235	sky blue
135 206 235	SkyBlue
135 206 255	SkyBlue1
126 192 238	SkyBlue2
108 166 205	SkyBlue3
74 112 139	SkyBlue4
106 90 205	slate blue
112 128 144	slate gray
112 128 144	slate grey
106 90 205	SlateBlue
131 111 255	SlateBlue1
122 103 238	SlateBlue2

RGB	Color Name
105 89 205	SlateBlue3
71 60 139	SlateBlue4
112 128 144	SlateGray
198 226 255	SlateGray1
185 211 238	SlateGray2
159 182 205	SlateGray3
108 123 139	SlateGray4
112 128 144	SlateGrey
255 250 250	snow
255 250 250	snow1
238 233 233	snow2
205 201 201	snow3
139 137 137	snow4
0 255 127	spring green
0 255 127	SpringGreen
0 255 127	SpringGreen1
0 238 118	SpringGreen2
0 205 102	SpringGreen3
0 139 69	SpringGreen4
70 130 180	steel blue
70 130 180	SteelBlue
99 184 255	SteelBlue1
92 172 238	SteelBlue2
79 148 205	SteelBlue3
54 100 139	SteelBlue4
210 180 140	tan
255 165 79	tan1
238 154 73	tan2

RGB	Color Name
205 133 63	tan3
139 90 43	tan4
216 191 216	thistle
255 225 255	thistle1
238 210 238	thistle2
205 181 205	thistle3
139 123 139	thistle4
255 99 71	tomato
255 99 71	tomato1
238 92 66	tomato2
205 79 57	tomato3
139 54 38	tomato4
64 224 208	turquoise
0 245 255	turquoise1
0 229 238	turquoise2
0 197 205	turquoise3
0 134 139	turquoise4
238 130 238	violet
208 32 144	violet red
208 32 144	VioletRed
255 62 150	VioletRed1
238 58 140	VioletRed2
205 50 120	VioletRed3
139 34 82	VioletRed4
245 222 179	wheat
255 231 186	wheat1
238 216 174	wheat2
205 186 150	wheat3

RGB	Color Name
139 126 102	wheat4
255 255 255	white
245 245 245	white smoke
245 245 245	WhiteSmoke
255 255 0	yellow
154 205 50	yellow green
255 255 0	yellow1
238 238 0	yellow2
205 205 0	yellow3
139 139 0	yellow4
154 205 50	YellowGreen

Index

Symbols

! for templates, 116
! operator, 22, 24
!= operator, 23
- (minus) operator, 24
for templates, 116
% for templates, 116
& in computes, 17
&& operator, 22, 24
* for templates, 116
,, 11
. (dot) operator, 24
< in computes, 17
< operator, 23
<= operator, 23
= operator, 24
== operator, 23
-> operator, 24
> operator, 23
>= operator, 23
? : operator, 22, 24
? for templates, 116
@ for templates, 116
[ ] operator, 24
[Display], 2
[Types], 2
^ operator, 23
|| operator, 22, 24

A

above modifier, 133
abs function, 279
absolute modifier, 133
absolute positioning, 131, 136
absolute value, determining, 279
accessibility messages, 86
acclabel option, 11, 86
acos function, 280
action item, 9, 52
    determining if the item is activated, 88
    determining the next item on the page, 140
    determining the previous item on the page, 141
    filtering submissions by datagroup, 225
    filtering submissions by group, 231
    filtering submissions by item reference, 233
    filtering submissions by item type, 235
    filtering transmissions by datagroup, 225
    filtering transmissions by group, 231
    filtering transmissions by item reference, 233
    filtering transmissions by item type, 235
    filtering transmissions by namespace, 237
    filtering transmissions by option reference, 239
    filtering transmissions by option type, 241
    filtering transmissions by page reference, 243
    linking data items, 96
    linking to a data item, 95
    setting a datagroup, 96
    setting an action to repeat, 98
    setting the destination URL, 251
    setting the printing options, 177
    setting the timing for an action, 98
    setting the transmission format, 227
    setting the type of action, 246
    setting the type of action triggered, 247
    setting the url for submissions, 249
    setting whether the item is active, 90
actions
    cancel, 246
    cells that trigger actions, 57
    display, 247
    done, 246
    enclose, 247
    extract, 247
    link, 246
    pagedone, 246
    print, 247
    refresh, 247
    remove, 247
    replace, 246
    save, 247
    select, 247
    signature, 247
    submit, 246

activated option, 11, 88
activating items, 90
active option, 11, 90
add_ds format flag, 113
additive operators, 23
after modifier, 133
alignb2b modifier, 134, 135
alignb2c modifier, 134
alignb2t modifier, 134
alignc2b modifier, 134
alignc2l modifier, 134
alignc2r modifier, 134
alignc2t modifier, 134
alignhorizbetween modifier, 134
alignhorizc2c modifier, 134
alignl2c modifier, 134
alignl2l modifier, 134
alignl2r modifier, 134
alignr2c modifier, 134
alignr2l modifier, 134
alignr2r modifier, 134
alignt2b modifier, 134
alignt2c modifier, 134
alignt2t modifier, 134
alignvertbetween modifier, 134
alignvertc2c modifier, 134
AM, removing from input, 116
angles
    converting degrees to radians, 288
    converting radians to degress, 297
    determining the arc cosine, 280
    determining the arc sine, 283
    determining the arc tangent, 284
    determining the cosine, 287
    determining the sine, 300
    determining the tangent, 302
annuity function, 281
appearance of forms, preventing changes, 39
application
    determining the name of, 303
    determining version of, 304, 305
application/vnd.xfdl, 5, 227
application/vnd.xfdl with compression, 227
application/x-www-form-urlencoded, 227
applicationName function, 303
applicationVersion function, 304
applicationVersionNum function, 305
arc cosine, determining, 280
arc sine, determining, 283
arc tangent, determining, 284
array elements
    element names, 84
    syntax, 83, 84
arrays
    definition, 16
    element references, 27
    elements and options, 11
    syntax for, 16
asin function, 283
assignment operator, 24
atan function, 284
attachment buttons, 54
attachments
    grouping, 96
    setting the datagroup, 54
    storing data for attachments, 63, 95
    storing the file name, 104
attachments *See* enclosures
attributes
    for certificates, 194
    getting the value of an attribute, 320
    setting the value of an attribute, 334
    version attribute, 7
automatic actions *See* action item

B

background color *See* color
Backus Naur Form *See* BNF
base, converting to base 10, 311
base64 encoding, 156
base64 encoding of binary data, 18, 156
base64-gzip encoding, 156
before modifier, 133
below modifier, 133
bgcolor option, 11, 91
binary data objects, 18
BNF rules
    for function calls, 257
    for XFDL, 2
bold, setting for font, 110

border width
  setting for print, 177
bordercolor option, 93
borders
  setting the border color, 93
  setting the width of the label border, 148
borderwidth option, 12
box item, 9, 53
  determining the next item on the page, 140
  determining the previous item on the page, 141
  positioning the item on the form, 132
  setting the background color, 91
  setting the background color for print, 169
  setting the border width, 93
  setting the font style, 110
  setting the size, 221
  setting whether the item is visible, 255
bracket_negative format flag, 113
bracketing negative values, 113
built-in labels, 145
buttom item
  setting the file format for saves, 182
button item, 9, 54
  adding accessibility messages, 86
  attachments
    creating attachment buttons, 54
  capturing the last keystroke (hotkeys), 143
  controlling the display of an image, 127
  controlling which certificate details the user sees, 193
  controlling which certificates are available to the user, 193
  creating submit buttons, 54
  detecting if a button is clicked, 336
  detecting whether the mouse is over the item, 160
  determining if the item is activated, 88
  determining the next item on the page, 140
  determining the previous item on the page, 141
  determining whether the item has the focus, 105
  enclosure buttons, 54
  filtering signatures by datagroup, 189
  filtering signatures by group, 207
  filtering signatures by item reference, 209
  filtering signatures by item type, 211
  filtering signatures by namespace, 213
filtering signatures by option reference, 215
filtering signatures by option type, 217
filtering signatures by page reference, 219, 243
filtering submissions by datagroup, 225
filtering submissions by group, 231
filtering submissions by item reference, 233
filtering submissions by item type, 235
filtering transmissions by datagroup, 225
filtering transmissions by group, 231
filtering transmissions by item reference, 233
filtering transmissions by item type, 235
filtering transmissions by namespace, 237
filtering transmissions by option reference, 239
filtering transmissions by option type, 241
filtering transmissions by page reference, 243
formatting the text, 112
justifying text, 142
linking data items, 96
linking to a data item, 95
linking to a help message, 125
linking to a signature image, 188
linking to a signature item, 187
linking to an image, 126
positioning the item on the form, 132
recording the location of a click, 94
recording the signer's identity, 196
setting a datagroup, 96
setting the background color, 91
setting the background color for print, 169
setting the border width, 93
setting the button's action, 246
setting the destination URL, 251
setting the font color, 108
setting the font color for print, 171
setting the font style, 110
setting the next item in the tab order, 162
setting the previous item in the tab order, 168
setting the printing options, 177
setting the signature MIME type, 198
setting the size, 221
setting the space between lines of text, 155
setting the text displayed, 253
setting the transmission format, 227
setting the type of action triggered, 247
setting the type of signature created, 187, 199

setting the url for submissions, 249
setting whether the item is active, 90
setting whether the item is visible, 255
signature buttons, 55
storing the identity of the signer, 196

C

caching references for computes, 34
calculations *See* computes
cancel action, 246
case
changing a value to lower case, 114
changing a value to title case, 114
changing a value to upper case, 114
ignoring case in edit checks, 116
making a string lower case, 274
making a string upper case, 275
case_insensitive check flag, 116
CDATA, 17
ceiling function, 285
cell item, 9, 57
cells that trigger actions, 57
dereferencing cells, 58
determining if the item is activated, 88
determining the next item on the page, 140
determining the previous item on the page, 141
filtering transmissions by datagroup, 225
filtering transmissions by group, 231
filtering transmissions by item reference, 233
filtering transmissions by item type, 235
filtering transmissions by namespace, 237
filtering transmissions by option reference, 239
filtering transmissions by option type, 241
filtering transmissions by page reference, 243
linking cells to other items, 123
linking to a data item, 95
linking to data items, 96
linking to lists, 123
select cells, 57
setting a datagroup, 96
setting the built-in label text, 145
setting the cell's action, 246
setting the destination URL, 251
setting the file format for saves, 182
setting the printing options, 177
setting the transmission format, 227
setting the type of action triggered, 247
setting the url for submissions, 249
setting the value, 253
setting whether the item is active, 90
centering text, 142
certificates
available attributes, 194
controlling which details are shown, 191
filtering which certificates are available, 191
setting which signature engine to use, 198
change list, duplicate entries, 35
changes, preventing form layout changes, 39
char data type, 85
character set
setting the character set for the label option, 151
character sets
setting the character set, 110
characters
locating in a string, 269
restricted, 17
check flags, 115
case_insensitive, 116
cross-reference table of, 117
fail_checks, 116
ignore_checks, 116
length, 116
mandatory, 116
message, 116
optional, 116
range, 116
strip_meridiem, 116
template, 116
check item, 9, 59
adding accessibility messages, 86
capturing the last keystroke (hotkeys), 143
detecting whether the mouse is over the item, 160
determining the next item on the page, 140
determining the previous item on the page, 141
determining whether the item has the focus, 105
linking to a help message, 125
positioning the item on the form, 132
setting the background color, 91
setting the background color for print, 169
setting the background color for the built-in label,

146
setting the border width for the built-in label, 148
setting the built-in label text, 145
setting the built-in label's background color for print, 173
setting the built-in label's font color for print, 175
setting the check to be on or off, 253
setting the font color, 108
setting the font color for print, 171
setting the font color for the built-in label, 149
setting the font style, 110
setting the font style for the built-in label, 151
setting the next item in the tab order, 162
setting the previous item in the tab order, 168
setting the size, 221
setting the state to readonly or writeonly, 101
setting whether the item is active, 90
setting whether the item is visible, 255
checklists, creating with check item, 59
checkValidFormats function, 306
child, determining which child an element is, 324
choices
presenting choices, 60, 70, 73, 75
setting choices to select, 247
choices, presenting choices, 57, 59
CIC signatures, 188
Click-wrap signature engine parameters, 204
clipping images, 127
closing a form, 246
color
list of colors, 387
setting the background color, 91
setting the background color for print, 169
setting the border color, 93
setting the font color, 108
setting the font color for print, 171
setting the label's background color, 146
setting the label's background color for print, 173
setting the label's font color, 149
setting the label's font color for print, 175
color data type, 85
combobox item, 9, 60
adding accessibility messages, 86
capturing the last keystroke (hotkeys), 143
detecting whether the mouse is over the item, 160
determining if the item is activated, 88
determining the next item on the page, 140
determining the previous item on the page, 141
determining whether the item has the focus, 105
formatting the text, 112
justifying text, 142
linking to a help message, 125
linking to cells, 123
positioning the item on the form, 132
setting the background color, 91
setting the background color for print, 169
setting the background color for the built-in label, 146
setting the border width for the built-in label, 148
setting the built-in label text, 145
setting the built-in label's background color for print, 173
setting the built-in label's font color for print, 175
setting the font color, 108
setting the font color for print, 171
setting the font color for the built-in label, 149
setting the font style, 110
setting the font style for the built-in label, 151
setting the next item in the tab order, 162
setting the previous item in the tab order, 168
setting the size, 221
setting the state to readonly or writeonly, 101
setting whether the item is visible, 255
setting which cell is selected, 253
comma_delimit format flag, 113
comparing strings, 267
compound function, 286
compute system, 21
computes
cached dependency lists, 30
conditional expressions, 22
duplicate entries on the total change list, 35
handling deleted form nodes, 35
handling dynamic references, 33
ignoring whitespace, 21
limitations, 35
mathematical expressions, 22
missing references, 35
re-entrancy, 34
reference caching, 34

representing computes, 29
restricted characters, 17
running computes, 29
topological sorting, 32
conditional expressions, 22
content format
syntax for an array, 16
syntax for options, 15
context sensitive help, 66, 125
coordinate data type, 85
coordinates option, 12, 94
copies, number printed, 178
copying form elements, 313
cos function, 287
cosine
determining the arc cosine, 280
determining the cosine, 287
countDatagrouptems function, 308
countGroupedItems function, 309
countLines function, 259
countWords function, 261
cross-reference
for data types, format flags, and check flags, 117
items and options, 353, 376
CryptoAPI signature engine parameters, 203
currency, adding a dollar sign to values, 113
current item, determining which item has the focus, 107
custom item, 82
naming conventions, 82
custom option, 256

D

daemon forms, 52
data item, 9, 63
base64 encoding, 18
determining the next item on the page, 140
determining the previous item on the page, 141
linking an image to a signature, 188
linking images to other items, 126
recording the filename of a data object, 104
setting a datagroup, 96
setting the MIME type, 159
storing the data, 156
storing the MIME data, 156
data option, 12, 95
data types, 2
char, 85
color, 85
coordinate, 85
cross-reference table, 117
date, 113
day_of_month, 113
day_of_week, 113
dollar, 113
float, 113
for fields, 112
for options, 85
integer, 85, 112
month, 113
setting for items, 112
string, 85, 112
time, 113
unsigned, 85
void, 113
year, 113
data, storing custom data in a form, 256
datagroup option, 12, 54, 96
filtering signatures by datagroup, 189
filtering submissions by datagroup, 225
datagroups, counting the items in a datagroup, 308
date
converting a date to seconds, 341
determining the current date, 349
determining the date, 340
determining the day, 343
determining the day of the week, 344
determining the last day of the month, 345
determining the month, 348
determining the year, 352
date data type, 113
date function, 340
date functions, 340
date, 340
dateToSeconds, 341
day, 343
dayOfWeek, 344
endOfMonth, 345
month, 348
year, 352
See also time functions

dateToSeconds function, 341
day
    determining the day, 340, 343
    determining the day of the week, 344
    determining the last day of the month, 345
day function, 343
day_of_month data type, 113
day_of_week data type, 113
dayOfWeek function, 344
deactivating items, 90
decimal function, 311
decision operators, 24
decisions, 22
default form settings, 47
default page settings, 49
default sizes for items, 379
deg2rad function, 288
degrees
    converting degrees to radians, 288
    converting radians to degrees, 297
delay option, 12, 98
deleting form elements, 312
deleting form nodes with computes, 35
delimiters
    comma, 113
    space, 113
dependency lists, computes, 30
dereferencing cells, 58
design goals, XFDL, 1
destroy function, 312
dirtyflag option, 12, 100
display action, 247
display elements
    bgcolor, 91
    bordercolor, 93
    boxes, 53
    fontcolor, 108
    itemlocation, 131
    justify, 142
    label, 145
    labelbgcolor, 146
    labelborderwidth, 148
    labelfontcolor, 149
    labelfontinfo, 151
    spacer, 80
    thickness, 224
display mechanisms, 2
displaying
    images, 67
    text, 67, 145
document reproducibility, 18
document-centric model, 1, 3
dollar data type, 113
dollar sign, adding to values, 113
done action, 246
duplicate function, 313
duplicating form elements, 313
dynamic references, 28
    in computes, 33

E

edit checks, 115
    checking all formatting, 306
    determining if a value has a valid format, 330
    finding invalid items, 306
    forcing failure, 116
    ignoring edit checks, 116
editstate option, 12, 101
effects, setting for font, 110
elements, 7
    deleting form elements, 312
    duplicating form elements, 313
    locating an element, 328
    naming elements, 18
    preventing exploitable overlap of signed elements, 40
    references to, 27
    root element, 7
    securing signed elements, 39
    setting the value of an element, 332
email, setting the url for email submissions, 249
enclose action, 247
enclosure data, 63
enclosures
    enclosure buttons, 54
    grouping, 96
    setting a display action, 247
    setting a remove action, 247
    setting an enclose action, 247
    setting an extract action, 247

setting the datagroup, 54
storing the file name, 104
enclosures, storing, 95
encoding
base64, 156
base64-gzip, 156
endOfMonth function, 345
engines, signature
parameters for common engines, 203
setting the signature engine to use, 198
Entrust signature engine parameters, 206
epilog, XML, 18
error checking, 112
error messages, setting, 116
escape sequence for labels, 67
excludedmetadata option, 12, 102
exp function, 289
expandb2c modifier, 135
expandb2t modifier, 135
expandl2c modifier, 135
expandl2l modifier, 135
expandl2r modifier, 135
expandr2c modifier, 135
expandr2l modifier, 135
expandr2r modifier, 135
expandt2b modifier, 135
expandt2c modifier, 135
expandt2t modifier, 135
exponentiation operators, 23
exponents
calculating an exponential value, 296
calculating the exponentiation, 289
extensions, compatible, 5
extent modifier, 135, 137
extract action, 247

F

fact function, 290
factorial, calculating, 290
fail_checks check flag, 116
field item, 9, 64
adding accessibility messages, 86
capturing the last keystroke (hotkeys), 143
counting lines of text, 259
counting the number of words, 261
creating rich text fields, 180, 223
detecting whether the mouse is over the item, 160
determining the next item on the page, 140
determining the previous item on the page, 141
determining whether the item has the focus, 105
formatting the text, 112
justifying text, 142
linking to a help message, 125
positioning the item on the form, 132
setting the background color, 91
setting the background color for print, 169
setting the background color for the built-in label, 146
setting the border width, 93
setting the border width for the built-in label, 148
setting the built-in label text, 145
setting the built-in label's background color for print, 173
setting the built-in label's font color for print, 175
setting the font color, 108
setting the font color for print, 171
setting the font color for the built-in label, 149
setting the font style, 110
setting the font style for the built-in label, 151
setting the horizontal scrolling, 185
setting the next item in the tab order, 162
setting the previous item in the tab order, 168
setting the size, 221
setting the state to readonly or writeonly, 101
setting the value, 253
setting the vertical scrolling, 186
setting whether the item is active, 90
setting whether the item is visible, 255
file formats, setting for saves, 182
file name, for enclosures, 104
filename option, 12, 104
filtering
general information for signatures, 37
order of precedence, 381
signatures by datagroup, 189
signatures by group, 207
signatures by item references, 209
signatures by item type, 211
signatures by namespace, 213
signatures by option references, 215

signatures by option type, 217
signatures by page reference, 219
submissions by datagroup, 225
submissions by group, 231
submissions by item reference, 233
submissions by item type, 235
submissions by option reference, 239
submissions by option type, 241
submissions by page reference, 243
transmissions by namespace, 237
triggeritem and coordinates, 56
which certificates are available, 191
flags
for changes in form status, 100
See also check flags
See also format flags
float data type, 113
floor function, 291
focus
determining whether an item has the focus, 105
determining which item has the focus, 107
focused option, 11, 105
focuseditem option, 12, 107
folders, for enclosures, 96
fontcolor, 11
fontcolor option, 11, 108
fontinfo option, 12, 110
fonts
setting font characteristics, 110
setting the character set, 110
setting the character set for the label option, 151
setting the color, 108
setting the label option's font, 151
form
element definition, 7
form globals, 47
page definition, 7
revision number, 120
serial number, 120
title for a form, 120
form global
adding a title, serial number, and version, 120
capturing the last keystroke (hotkeys), 143
detecting whether the form has changed, 100
determining if the form is activated, 88
determining whether the form has the focus, 105
recording which item triggered an event, 245
setting the default background color for built-in labels, 146
setting the font style for built-in labels, 151
setting the form's background color, 91
setting the form's background color for print, 169
setting the form's border width, 93
setting the form's default file format for saves, 182
setting the form's default font color, 108
setting the form's default font color for built-in labels, 149
setting the form's default font color for print, 171
setting the form's default font style, 110
setting the form's default printing options, 177
setting the form's default transmission format, 227
setting the title of a form, 145
format flags, 113
add_ds, 113
bracket_negative, 113
comma_delimit, 113
cross reference table, 117
long, 114
lower_case, 114
numeric, 115
short, 114
space_delimit, 113
title_case, 114
upper_case, 114
void, 115
format option, 12, 112
formatString function, 315
formatting
checking all formatting, 306
determining if a value has a valid format, 330
formatting, finding invalid items, 306
formid option, 12, 120
forms
daemon forms, 52
deleting form elements, 312
detecting a change in status, 100
duplicating form elements, 313
form names, 5
form structure, 7
locating a form element, 328

formulas *See* computes
fullname option, 122
function calls, 257
    passing literals and option references, 258
    syntax, 29
    See also date functions
    See also math functions
    See also string functions
    See also time functions
    See also utility functions

G

generateUniqueName function, 317
get function, 319
getAttr function, 320
getGroupedItem function, 322
getPosition function, 324
getPref function, 325
getReference function, 328
globals
    form globals, 47
    page globals, 49
graphics *See* images
group option, 13, 123
    filtering signatures by group, 207
groups
    counting the items in a group, 309
    filtering submissions by group, 231
    getting the sid of a grouped item, 322
    grouping attachments, 96
    grouping cells, 123
    grouping enclosures, 96
    grouping radio buttons, 123

H

height, setting for items, 221
help item, 9, 66
    determining the next item on the page, 140
    determining the previous item on the page, 141
    linking to other items, 125
    setting the help message, 253
    setting whether the item is active, 90
help messages, 66, 125
help option, 13, 125
hiding items, 255
HMAC-ClickWrap signature engine parameters, 204
HMAC-ClickWrap signatures, 102
horizontal scrolling in fields, 185
hour
    determing the time, 351
    determining the hour, 346
hour function, 346
HTML
    submission format, 228
    submitting forms as, 227

I

identifiers, scope, 18
ignore_checks check flag, 116
image data, 63
image option, 13, 126
imagemode option, 13, 127
images
    controlling the display of, 127
    displaying, 67
    linking an image to a signature, 188
    linking items to images, 126
    mapping, 94
    setting the MIME type, 159
    storing, 95
impairments, messages for sight impairments, 86
index
    getting the position index of an element, 324
    locating a character in a string, 269
    locating a substring within a string, 270, 271
indirect references, 27, 58
InkTools signature engine parameters, 203
input
    capturing the last keystroke, 143
    forcing edit check failure, 116
    ignoring edit checks, 116
    mandatory, 112
    reformatting, 112
    removing the AM or PM, 116
    restricting, 112
    setting a length, 116
    setting a range, 116
    setting a template, 116
    setting input error message, 116

setting input to be mandatory, 116
setting input to be optional, 116
input fields, 64
integer data type, 85, 112
interest, determining compound interest, 286
invisible items, 255
isValidFormat function, 330
italics, setting for font, 110
item details
definition, 8
scope identifier, 18
tag, 18
item references
filtering signatures by item references, 209
filtering submissions by item reference, 233
item tags, relative, 129, 130, 140, 141
itemfirst option, 129
itemlast option, 130
itemlocation option, 13, 131
itemnext option, 140
itemprevious option, 141
items
action, 9, 52
box, 9, 53
button, 9, 54
cell, 9, 57
check, 9, 59
combobox, 9, 60
cross referenced with options, 353, 376
custom, 82
data, 9, 63
default sizes, 379
deleting form elements, 312
determining if item is activated, 88
determining if the mouse is over an item, 160
determining the first item on a page, 129
determining the last item on a page, 130
determining the next item on the page, 140
determining the previous item on the page, 141
determining the tab order, 162
duplicating form elements, 313
field, 9, 64
filtering signatures by item type, 211
filtering submissions by item type, 235
general details, 51
help, 9, 66
label, 9, 67
line, 10, 69
list, 10, 70
locating a form element, 328
making items invisible, 255
popup, 10, 73
positioning items on the form, 131
preventing exploitable overlaps of signed items, 40
radio, 10, 75
referencing, 27
setting the size of, 221
setting the tab order, 168
signature, 10, 77
sizing items, 137
spacer, 10
toolbar, 10, 81
turning items off, 90

J

justify option, 13, 142
justifying text, 142

K

keypress option, 13, 143
keystroke, capturing the last, 143

L

label item, 9, 67
adding line breaks, 67
controlling the display of an image, 127
determining the next item on the page, 140
determining the previous item on the page, 141
formatting the text, 112
justifying text, 142
linking to a help message, 125
linking to an image, 126
positioning the item on the form, 132
setting the background color, 91
setting the background color for print, 169
setting the border width, 93
setting the font color, 108
setting the font color for print, 171
setting the font style, 110

setting the size, 221
setting the space between lines of text, 155
setting the text displayed, 253
setting whether the item is active, 90
setting whether the item is visible, 255
label option, 13, 145
setting a background color, 146
setting a font, 151
setting a font color, 149
setting the width of the label border, 148
labelbgcolor option, 11, 146
labelborderwidth option, 12, 148
labelfontcolor option, 11, 149
labelfontinfo option, 12, 151
landscape printing, 178
layout, preventing changes, 39
layout, storing the layout information, 153
layoutinfo option, 153
leading, adjusting the space between lines of text, 155
left justify, 142
length
determing for strings, 266
setting for input, 116
length check flag, 116
lexical constraints in top level structure, 7
line breaks in labels, 67
line item, 10, 69
determining the next item on the page, 140
determining the previous item on the page, 141
positioning the item on the form, 132
setting the font color for print, 171
setting the font style, 110
setting the line color, 108
setting the size, 221
setting the thickness, 224
setting whether the item is visible, 255
lines, counting lines in a field, 259
linespacing option, 13, 155
link action, 246
linking cells to lists, 123
list item, 10, 70
capturing the last keystroke (hotkeys), 143
detecting whether the mouse is over the item, 160
determining the next item on the page, 140
determining the previous item on the page, 141
determining whether the item has the focus, 105
formatting the text, 112
linking to a help message, 125
linking to cells, 123
positioning the item on the form, 132
setting the background color, 91
setting the background color for print, 169
setting the background color for the built-in label, 146
setting the border width, 93
setting the border width for the built-in label, 148
setting the built-in label text, 145
setting the built-in label's background color for print, 173
setting the built-in label's font color for print, 175
setting the font color, 108
setting the font color for print, 171
setting the font color for the built-in label, 149
setting the font style, 110
setting the font style for the built-in label, 151
setting the next item in the tab order, 162
setting the size, 221
setting the state to readonly or writeonly, 101
setting whether the item is active, 90
setting whether the item is visible, 255
setting which cell is selected, 253
ln function, 292
location
locating a substring in a string, 271
location of items on the form, 131
log function, 293
logarithm
determining the logarithm, 293
determining the natural logarithm, 292
logic, 22
logical and, 22
logical negation, 22
logical operators, 24
logical or, 22
long format flag, 114
lower case, changing a value to, 114
lower case, making a string, 274
lower_case format flag, 114

M

mandatory check flag, 116
mandatory input, 116
    setting, 112
mapping images, 94
math functions, 279
    abs, 279
    acos, 280
    annuity, 281
    asin, 283
    atan, 284
    ceiling, 285
    compound, 286
    cos, 287
    deg2rad, 288
    exp, 289
    fact, 290
    floor, 291
    ln, 292
    log, 293
    mod, 294
    pi, 295
    power, 296
    rad2deg, 297
    rand, 298
    round, 299
    sin, 300
    sqrt, 301
    tan, 302
mathematical expressions, 22
mechanisms, display, 2
membership
    array membership, 24
    indirect membership, 24
    structure membership, 24
membership operators, 24
meridiem, removing from input, 116
message check flag, 116
messages, help, 66
methods *See* functions
MIME type, 159
MIME types, 5
    application/vnd.xfdl, 227
    application/vnd.xfdl with compression, 227
    application/x-www-form-urlencoded, 227
mimedata option, 13, 156
    usage, 18
mimetype option, 12, 159
minute
    determining the minute, 347
    determining the time, 351
minute function, 347
mod function, 294
modifiers
    above, 133
    absolute, 133
    after, 133
    alignb2b, 134, 135
    alignb2c, 134
    alignb2t, 134
    alignc2b, 134
    alignc2l, 134
    alignc2r, 134
    alignc2t, 134
    alignhorizbetween, 134
    alignhorizc2c, 134
    alignl2c, 134
    alignl2l, 134
    alignl2r, 134
    alignr2c, 134
    alignr2l, 134
    alignr2r, 134
    alignt2b, 134
    alignt2c, 134
    alignt2t, 134
    alignvertbetween, 134
    alignvertc2c, 134
    before, 133
    below, 133
    expandb2c, 135
    expandb2t, 135
    expandl2c, 135
    expandl2l, 135
    expandl2r, 135
    expandr2c, 135
    expandr2l, 135
    expandr2r, 135
    expandt2b, 135
    expandt2c, 135

expandt2t, 135
extent, 135, 137
offset, 133
position, 133
within, 133
modulus, determining the modulus, 294
month
determining the last day of the month, 345
determining the month, 340, 348
month data type, 113
month function, 348
mortgage
calculating annuity, 281
determining compound interest, 286
mouse, tracking the location of, 160
mouseover option, 11, 160
multiplicative operators, 23

N

name of enclosed file, 104
name of signer, 122
names
compatible extensions, 5
form name, 5
generating unique, 317
namespaces
filtering signatures by namespace, 213
filtering transmissions by namespace, 237
naming elements, 18
natural logarithm, determining, 292
negative values, bracketing, 113
Netscape signature engine parameters, 204
next option, 13, 162
non-repudiation, 1, 3, 153
notarizing signatures, 102
notation, XFDL, 2
now function, 349
numbers, generating random, 298
numeric format flag, 115

O

off, turning items off, 90
offset for space between lines of text, 155
offset modifier, 133
operations, order of operations, 22, 24
operators
additive, 23
assignment, 24
decision, 24
exponentiation, 23
logical, 24
membership, 24
multiplicative, 23
relational, 23
table of operators, 23
unary minus, 24
option
getting the value of an option, 319
option details
definition, 11
option references
filtering signatures by option references, 215
filtering submissions by option reference, 239
optional check flag, 116
options
acclabel, 11, 86
activated, 11, 88
active, 11, 90
bgcolor, 11, 91
bordercolor, 93
borderwidth, 12
content format, 15
coordinates, 12, 94
cross-referenced with items, 353, 376
custom, 256
data, 12, 95
data types, 85
datagroup, 12, 54, 96
delay, 12, 98
dirtyflag, 12
dirtyflag option, 100
editstate, 12, 101
excludedmetadata, 12, 102
filename, 12, 104
filtering signatures by option type, 217
filtering submissions by option type, 241
focused, 11, 105
focuseditem, 12, 107
fontcolor, 11, 108

fontinfo, 12, 110
format, 12, 112
formid, 12, 120
fullname, 122
getting the value of an option, 319
group, 13, 123
help, 13, 125
image, 13, 126
imagemode, 13, 127
itemfirst, 129
itemlast, 130
itemlocation, 13, 131
itemnext, 140
itemprevious, 141
justify, 13, 142
keypress, 13, 143
label, 13, 145
labelbgcolor, 11, 146
labelborderwidth, 12, 148
labelfontcolor, 11, 149
labelfontinfo, 12, 151
layoutinfo, 153
linespacing, 13, 155
locating a form element, 328
mimedata, 13, 156
mimetype, 12, 159
mouseover, 11, 160
next, 13, 162
order of precedence, 84
pagefirst, 163
pageid, 14, 164
pagelast, 165
pagenext, 166
pageprevious, 167
previous, 13, 168
printbgcolor, 14, 169
printfontcolor, 14, 171
printlabelbgcolor, 14, 173
printlabelfontcolor, 14, 175
printsetting, 177
printsettings, 14
quick reference table, 356
rtf, 180
saveformat, 14, 182
scrollhoriz, 14, 185
scrollvert, 14, 186
setting the value of an option, 332
signature, 187
signatureimage, 188
signdatagroups, 189
signdetails, 191
signer, 196
signformat, 198
signgroups, 207
signitemrefs, 209
signitems, 211
signnamespaces, 213
signoptionrefs, 215
signoptions, 217
signpagerefs, 219
size, 13, 221
textype, 223
thickness, 13, 224
transmitdatagroups, 14, 225
transmitformat, 14, 227
transmitgroups, 14, 231
transmititemrefs, 14, 233
transmititems, 14, 235
transmitnamespaces, 237
transmitoptionrefs, 14, 239
transmitoptions, 14, 241
transmitpagerefs, 14, 243
triggeritem, 14, 245
type, 14, 246
url, 14, 249
value, 15, 253
version, 15
version, referencing, 29
visible, 15, 255
options, syntax, 83
order of operations, 22, 24
order of precedence, options, 84
orientation
for printing, 178
origins
UFDL, 1
XFDL, 1
overlapping signatures, 39

P

pad function, 262
padding strings, 262
page
  definition, 7
  determining the first page in a form, 163
  determining the last page in a form, 165
  determining the next page in the form, 166
  determining the previous page in the form, 167
page global
  capturing the last keystroke (hotkeys), 143
  detecting whether the mouse is over the page, 160
  determining if the page is activated, 88
  determining the first item on a page, 129
  determining the first page in a form, 163
  determining the last item on a page, 130
  determining the last page in a form, 165
  determining the next page in the form, 166
  determining the previous page in the form, 167
  determining whether a page has the focus, 105
  determining which item has the focus, 107
  setting the default background color for built-in labels, 146
  setting the first item in the tab order, 162
  setting the font style for built-in labels, 151
  setting the page background color, 91
  setting the page's background color for print, 169
  setting the page's border width, 93
  setting the page's default file format for saves, 182
  setting the page's default font color, 108
  setting the page's default font color for built-in labels, 149
  setting the page's default font color for print, 171
  setting the page's default font style, 110
  setting the page's default printing options, 177
  setting the page's default transmission format, 227
  setting the previous item in the tab order, 168
  setting the serial number for a page, 164
  setting the title of a page, 145
page globals, 49
page references
  filtering signatures by page reference, 219
  filtering submissions by page reference, 243
page tags, relative, 163, 165, 166, 167
pagedone action, 246
pagefirst option, 163
pageid option, 14, 164
pagelast option, 165
pagenext option, 166
pageprevious option, 167
pages
  flipping pages, 246
  setting which pages print, 177
paging controls *See* button item
panels *See* box item
paramaters
  passing literals and options references, 258
parameters
  for common signature engines, 203
pi function, 295
placement of items on the form, 131
PM, removing from input, 116
point size, setting, 110
popup item, 10, 73
  adding accessibility messages, 86
  capturing the last keystroke (hotkeys), 143
  detecting whether the mouse is over the item, 160
  determining if the item is activated, 88
  determining the next item on the page, 140
  determining the previous item on the page, 141
  determining whether the item has the focus, 105
  formatting the text, 112
  justifying text, 142
  linking to a help message, 125
  linking to cells, 123
  positioning the item on the form, 132
  setting the background color, 91
  setting the background color for print, 169
  setting the border width, 93
  setting the built-in label text, 145
  setting the font color, 108
  setting the font color for print, 171
  setting the font style, 110
  setting the next item in the tab order, 162
  setting the previous item in the tab order, 168
  setting the size, 221
  setting the state to readonly or writeonly, 101
  setting whether the item is active, 90
  setting whether the item is visible, 255

setting which cell is selected, 253
portrait printing, 178
position
determing an element's position within its parent, 324
locating a character in a string, 269
position modifiers, 133
positioning items, 131
power function, 296
precedence
for filters, 381
of operations, 24
preferences, getting the user's Viewer preferences, 325
previous option, 13, 168
print action, 247
printbgcolor option, 14, 169
printfontcolor option, 14, 171
printing
setting a print action, 247
setting the background color, 169
setting the font color, 171
setting the label's background color, 173
setting the label's font color, 175
setting the number of copies, 178
setting the orientation, 178
setting the print options, 177
printlabelbgcolor option, 14, 173
printlabelfontcolor option, 14, 175
printsettings option, 14, 177
prolog, XML, 18
prompting to save a form, 100

Q

quick reference
default item sizes, 379
for options, 356
list of colors, 387
order of precedence for filters, 381
quoted strings, 26

R

rad2deg function, 297
radians
converting degrees to radians, 288
converting radians to degrees, 297
radio item, 10, 75
adding accessibility messages, 86
capturing the last keystroke (hotkeys), 143
detecting whether the mouse is over the item, 160
determining the next item on the page, 140
determining the previous item on the page, 141
determining whether the item has the focus, 105
grouping, 123
linking to a help message, 125
linking to cells, 123
positioning the item on the form, 132
setting the background color, 91
setting the background color for print, 169
setting the background color for the built-in label, 146
setting the border width for the built-in label, 148
setting the built-in label text, 145
setting the built-in label's background color for print, 173
setting the built-in label's font color for print, 175
setting the font color, 108
setting the font color for print, 171
setting the font color for the built-in label, 149
setting the font style, 110
setting the font style for the built-in label, 151
setting the next item in the tab order, 162
setting the previous item in the tab order, 168
setting the radio to be on or off, 253
setting the size, 221
setting the state to readonly or writeonly, 101
setting whether the item is active, 90
setting whether the item is visible, 255
rand function, 298
random numbers, generating, 298
range check flag, 116
range, setting for input, 116
readers, messages for screen readers, 86
readonly, setting items to be, 101
readwrite, setting items to be, 101
re-entrancy for computes, 34
references
caching references for computes, 34
dereferencing cells, 58
dynamic, 28

dynamic references in computes, 33
element references, 27
filtering signatures by item reference, 209
filtering signatures by option reference, 215
filtering signatures by page reference, 219, 243
filtering submissions by item reference, 233
filtering submissions by item type, 235
indirect, 58
missing references in computes, 35
referencing the XFDL version, 29
static references, 27
references, source material, 2
reformatting input, 112
refresh action, 247
relational operators, 23
relative item tags, 129, 130, 140, 141
relative page tags, 163, 165, 166, 167
relative positioning, 131, 136
remainder, determining, 294
remove action, 247
repeating an action, 98
replace action, 246
replace function, 264
reproducibility of documents, 18
resizing images, 127
restricted characters in XFDL, 17
restricting input, 112
revision, number for forms, 120
rich text, creating rich text fields, 180, 223
right justify, 142
root element, 7
root, calculating square roots, 301
round function, 299
rounding
determining the ceiling, 285
determining the floor, 291
rounding a number, 299
rtf option, 180

S

save action, 247
save prompt, 100
saveformat option, 14, 182
saving forms, 247
setting the format, 182
scaling images, 127
scope identifiers, 18
screen readers, messages for, 86
scrollhoriz option, 14, 185
scrollvert option, 14, 186
second
converting a date to seconds, 341
converting seconds to a date, 340
determining the second, 350
second function, 350
securing signed elements, 39
select action, 247
serial number
for a form, 120
for a page, 164
set function, 332
setAttr function, 334
settings
global form settings, 47
global page settings, 49
short format flag, 114
shortcut keys, creating, 143
showing
images, 67
text, 67
sid, 18
generating unique sids, 317
getting the sid of a grouped item, 322
siganture item
controlling which certificates are available to the user, 193
sight impairments, messages for, 86
sign, adding a dollar sign to values, 113
signature action, 247
signature engines, parameters for common engines, 203
signature item, 10, 77
controlling which certificate details the user sees, 193
determining the next item on the page, 140
determining the previous item on the page, 141
filtering signatures by datagroup, 189
filtering signatures by group, 207
filtering signatures by item reference, 209
filtering signatures by item type, 211
filtering signatures by namespace, 213

filtering signatures by option reference, 215
filtering signatures by option type, 217
filtering signatures by page reference, 219, 243
filtering transmissions by namespace, 237
linking an image to a signature, 188
linking to a button item, 187
recording the identity of the signer, 122
recording the signature type, 187, 199
recording the signer's identity, 196
setting the MIME type, 198
storing signature notarizations, 102
storing the identity of the signer, 196
storing the layout information, 153
storing the layout information for the form, 153
storing the MIME data, 156
storing the signature, 156
signature option, 187
signatureimage option, 188
signatures
document-centric model, 3
failure conditions, 79
filtering, 79, 189, 207, 209, 211, 213, 215, 217, 219, 237
filtering triggeritem and coordinates, 56
HMAC-ClickWrap signatures, 102
in XFDL, 37
linking a button to a signature item, 187
linking an image to a signature, 188
mandatory signatures, 55
multiple signatures, 39
non-repudiation, 3
notarizing signatures, 102
order of precedence for filters, 381
overlapping signatures, 39
preventing exploitable overlaps of signed items, 40
preventing layout changes, 39
securing signed elements, 39
setting a signing action, 247
setting which certificate attributes are shown, 191
setting which signature engine to use, 198
signature buttons, 55
signature filters, 37
signer's name, 122
storing identity of signer, 196
storing the layout information, 153
signdatagroups option, 189
signdetails option, 191
signer option, 196
signer's name, 122
signformat option, 198
signgroups option, 207
signing forms, 77
signitemrefs option, 209
signitems option, 211
signnamespaces option, 213
signoptionrefs option, 215
signoptions option, 217
signpagerefs option, 219
sin function, 300
sine
determining the arc sine, 283
determining the sine, 300
size option, 13, 221
overriding with itemlocation, 222
sizes, defaults for items, 379
sizing an item, 137
sorting, topological for computes, 32
source material references, 2
space_delimit format flag, 113
spacer item, 10, 80
determining the next item on the page, 140
determining the previous item on the page, 141
positioning the item on the form, 132
setting the built-in label text, 145
setting the font style, 110
setting the size, 221
setting the space between lines of text, 155
spaces
removing leading and trailing whitespace from strings, 276
whitespace in computes, 21
spacing items on a form, 80
specifications, overlap with XFDL, 2
sqrt function, 301
square root, calculating, 301
status, detecting a change in the forms status, 100
string data type, 85, 112
string functions, 259
countLines, 259
countWords, 261

pad, 262
replace, 264
strlen, 266
strmatch, 267
strpbrk, 269
strrstr, 270
strstr, 271
substr, 272
tolower, 274
toupper, 275
trim, 276
URLDecode, 277
URLEncode, 278
strings
counting the number of lines, 259
counting the number of words, 261
decoding a URL encoded string, 277
determining the length of, 266
encoding a URL string, 278
formatting using set rules, 315
locating a character in a string, 269
locating a substring within a string, 270, 271
making a string lower case, 274
making a string upper case, 275
matching strings, 267
padding or truncating strings, 262
quoted strings, 26
removing leading and trailing whitespace, 276
replacing substrings, 264
retrieving a substring, 272
strip_meridiem check flag, 116
strlen function, 266
strmatch function, 267
strpbrk function, 269
strrstr function, 270
strstr function, 271
structure of XFDL
arrays, 11
binary data objects, 18
document reproducibility, 18
items, 8
options, 11
restricted characters, 17
scope identifiers, 18
top level structure of XFDL forms, 7
submissions
determining which item triggered a submission, 245
filtering by datagroup, 225
filtering by group, 231
filtering by item reference, 233
filtering by item type, 235
filtering by option reference, 239
filtering by option type, 241
filtering by page reference, 243
HTML syntax, 228
linking to another file, 246
order of precedence for filters, 381
setting a submit action, 246
setting the format of a submission, 227
setting the url for, 249
submit action, 246
submitting forms, 54
substr function, 272
substrings
locating a substring within a string, 270
replacing, 264
retrieving a substring, 272

T

tab order
changing by setting the focus, 105
setting the next item, 162
setting the previous item, 168
tags
generating unique tags, 317
tan function, 302
tangent
determining the arc tangent, 284
determining the tangent, 302
template check flag, 116
templates for input, 116
terminology, XFDL, 2
ternary operator, 22
text
adjusting the space between lines, 155
counting lines of text, 259
counting the number of words, 261
creating rich text fields, 180, 223
determining the length of, 266
displaying, 67, 145

formatting using set rules, 315
justifying, 142
locating a character in, 269
locating a string within a substring, 270, 271
making text lower case, 274
making text upper case, 275
matching text, 267
padding or truncating strings, 262
removing leading and trailing whitespace, 276
replacing substrings, 264
retrieving a substring, 272
textype option, 223
thickness option, 13, 224
time
determing the time, 351
determining the current time, 349
determining the hour, 346
determining the minute, 347
determining the second, 350
removing the AM or PM, 116
time data type, 113
time function, 351
time functions, 340
dateToSeconds, 341
hour, 346
minute, 347
now, 349
second, 350
time, 351
See also date functions
timing an action, 98
title case, changing a value to, 114
title, for a form, 120
title_case format flag, 114
titles, displaying, 67
toggle function, 336
tolower function, 274
toolbar item, 10, 81
detecting whether the mouse is over the item, 160
determining the next item on the page, 140
determining the previous item on the page, 141
moving items to the toolbar, 81
setting the background color, 91
toolbars, adding to a form, 81
topological sorting, computes, 32
toupper function, 275
transmissions *See* submissions
transmitdatagroups option, 14, 225
transmitformat option, 14, 227
transmitgroups option, 14, 231
transmititemrefs option, 14, 233
transmititems option, 14, 235
transmitnamespaces option, 237
transmitoptionrefs option, 14, 239
transmitoptions option, 14, 241
transmitpagerefs option, 14, 243
trigger, detecting if a value changes, 336
triggering actions
automatically, 52
from a list, 57, 60, 70, 73
with a button, 54
triggeritem option, 14, 245
trigonometry
determining the arc cosine, 280
determining the arc sine, 283
determining the arc tangent, 284
determining the cosine, 287
determining the sine, 300
determining the tangent, 302
trim function, 276
truncating strings, 262
turning items off, 90
type option, 14, 246
type settings, 112

U

UFDL
origins, 1
unary minus operator, 24
underline, setting for font, 110
universal form settings, 47
universal page settings, 49
unsigned data type, 85
upper case, changing a value to, 114
upper case, making a string, 275
upper_case format flag, 114
URL encoding
decoding a string, 277
encoding a string, 278
url option, 14, 249

URLDecode function, 277
URLEncode function, 278
user controls, 54
utility functions, 303
    applicationName, 303
    applicationVersion, 304
    applicationVersionNum, 305
    checkValidFormats, 306
    countDatagrouptems, 308
    countGroupedItems, 309
    decimal, 311
    destroy, 312
    duplicate, 313
    formatString, 315
    generateUniqueName, 317
    get, 319
    getAttr, 320
    getGroupedItem, 322
    getPosition, 324
    getPref, 325
    getReference, 328
    isValidFormat, 330
    set, 332
    setAttr, 334
    toggle, 336

V

value
    detecting if a value changes, 336
    determining an absolute value, 279
value option, 15, 253
    definition, 25
version
    determing version of running application, 304, 305
    form version, 120
version attribute, 7
version option, 15
version, referencing the XFDL version option, 29
vertical scrolling in fields, 186
Viewer, getting the user preferences, 325
visible option, 15, 255
vision impairments, messages for, 86
void data type, 113
void format flag, 115

W

week, determining the day of the week, 344
weight, setting for font, 110
whitespace
    removing leading and trailing whitespace from strings, 276
whitespace in computes, 21
width
    setting for items, 221
    setting for lines, 224
within modifier, 133
words, counting the number of, 261
writeonly, setting items to be, 101

X

XFDL
    compute system, 21
    design goals, 1
    element definition, 7
    element references, 27
    form structure, 7
    introduction to, 1
    item definition, 8
    MIME types, 5
    notation, 2
    overlap with other specifications, 2
    page definition, 7
    quoted strings, 26
    restricted characters, 17
    signatures in XFDL, 37
    submitting forms as, 227
    terminology, 2
    version, 7
XFDL, saving forms as, 182
XML epilog, 18
XML prolog, 18
x-y coordinates, 94

Y

year
    determining the year, 352
year data type, 113
year function, 352
year, determining the year, 340