NOTE-XFDL-19980902

Extensible Forms Description Language (XFDL) 4.0

W3C Note, September 2, 1998

Latest Version
http://www.w3.org/TR/NOTE-XFDL
This version
http://www.w3.org/TR/1998/NOTE-XFDL-19980902

Editors
John Boyer (UWI.Com) *(jboyer@uwi.com)*
Tim Bray (Textuality) *(tbray@textuality.com)*
Maureen Gordon (UWI.Com) *(mgordon@uwi.com)*

Status of this document

This document is the initial draft of the specification of the XFDL facility. It is intended for review and comment and is subject to change.

This document is a NOTE made available by the W3C for discussion only. This indicates no endorsement of its content, not that the Consortium has, is or will be allocating resources to the issues addressed by the NOTE.

This document is a Submission to W3C from UWI Unisoft Wares Inc. Please see Acknowledged Submissions to W3C regarding its disposition.

Abstract

This document describes an XML syntax for the Extensible Forms Description Language (XFDL). The purpose of XFDL is to solve the body of problems associated with digitally representing complex forms such as those found in business and government. The requirements include support for high precision layout, supporting documentation, integrated computations and input validation, multiple overlapping digital signatures, and legally binding auditable transaction records, by maintaining the whole form as a single unit such that digital signatures can capture the entire context of transactions.

Table of Contents

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, UWI.Com 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
2. is a 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 digital signature support, including
    a. capture of the whole context of a business transaction
    b. multiple signers
    c. different signers of (possibly overlapping) portions of a form
    d. freezing computations on signed portions of a form

This version of the XFDL specification may be distributed freely, as long as all text and notices remain intact.

1.2 References

[1] Boyer, J. Lexical and Syntactic Specification for the Universal Forms Description Language (UFDL) Version 4.0. UWI.Com – The Internet Forms Company. 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-19980110. 10 FEB 1998.

[3] Gordon, M. (Ed.) UFDL v4.0.1 Specification. UWI.Com – The Internet Forms Company. 1993-1998.

1.3 Terminology

Terms are defined in Section 1.2 of the XML specification [2].

1.4 Notation

XFDL instances 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 element contents and 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" because XFDL allows the insertion of arbitrary non-XFDL elements in certain well-defined places. Furthermore, the allowed content of several XFDL elements depends on the value of a particular attribute; once again, neither of these facts is 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 stylesheet facility; however, it is not clear which one should be used for this purpose. In this specification, all elements and attributes that are candidates for replacement by a standardized style mechanism are marked [Display].

Similarly, it is almost certainly the case that XFDL's data typing controls can and should be replaced by a W3C-standardized set of data type specifiers when one becomes available. In this specification, all elements and attributes that are candidates for replacement by standardized data type specifiers are marked [Types].

2. The Structure of XFDL Forms

2.1 Top-Level Structure

An XFDL form is an XML document whose root element type is *XFDL*. The root element has a required *version* attribute, which is a numeric dotted triplet consisting of the major, minor, and maintenance versions of the XFDL to which the element content conforms. The XFDL element may also have a *sid* attribute; the *sid* attribute gives a scope identifier, which is discussed in Section 2.5.

[1] <!ATTLIST XFDL version CDATA #REQUIRED>

[2] <!ATTLIST XFDL sid CDATA #IMPLIED>

Here are the lexical constraints of the values of the version and sid attributes:

[3] versionAttrValue ::= Digit+ '.' Digit+ '.' Digit+

[4] sid ::= Letter (Letter | Digit | '_')*

An XFDL element contains zero or more *option* elements followed by one or more *page* elements. The option elements that occur before the first page 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.

[5] <!ELEMENT XFDL (%options;*, page+)>

A *page* element contains zero or more *page global options* followed by zero or more item elements. Page global options typically contain information applicable to the whole page or default settings for options appearing within element content of items, and they take precedence over form global options. A page is also required to have a *sid* attribute.

[6] <!ELEMENT page (%options;*, %items;*)>

[7] <!ATTLIST page sid CDATA #REQUIRED>

The intention of using of 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, checkboxes, popup lists, and text fields. Other items are used to carry information such as an enclosed word processing document, a digital signature, daemon client-side actions, or application-specific job descriptions (such as workflow or ODBC 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 is required to have a *sid* attribute.

[8] <!ELEMENT %items; (%options;*)>

[9] <!ATTLIST %items; sid CDATA #REQUIRED>

The parameter entity reference to "%item;" could be defined partially as

[10] <!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 rule 10 are discussed in Section 5 and summarized here for convenience.

action
: a non-visible item that can perform the same task as a button (print, cancel, submit, etc.) 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 because a box is drawn under all widgets on a page.

button
: can be defined to perform 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, etc. 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 the actions of a button, such as submission, print, save, etc.

check
: defines a single checkbox.

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

data
: is used to carry binary information using base-64 encoding, such as enclosed files or digital images.

field
: is 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.

line
: a simple graphic effect used as a separator.

list
: shows a list box populated with cell items from which the user can choose.

popup
: shows either the text of the currently selected cell or its 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 digital signature that ultimately results when a user presses a signature button.

spacer
: an invisible GUI widget that facilitates relational positioning.

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.

This is only a partial list of items. XFDL allows application-defined items in the form. Simple, static application-specific information can be expressed using XML processing instructions, but many server side applications for workflow and ODBC require complex instructions that can include the use of the XFDL compute system to collect information from around the form.

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 attribute of an item, page, or form. The parameter entity reference to "%option;" could partially be defined as follows:

```
[11] <!ENTITY % option "(activated | active | bgcolor | borderwidth | coordinates | data | datagroup |
delay | editstate | filename | focused | fontcolor | fontinfo | format | group | help | image | itemlocation
justify | label | labelbgcolor | labelborderwidth | labelfontcolor | labelfontinfo | mimedata | mimetype |
mouseover | next | previous | printsettings | saveformat | scrollhoriz | scrollvert | signature |
signdatagroups | signer | signformat | signgroups | signitemrefs | signitems | signoptionrefs |
signoptions | size | thickness | transmitformat | triggeritem | type | url | value | visible)" >
```

Again, the definition is partial because XFDL also supports application-defined options. 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.

The options are fully discussed in section 6 and summarized here:

activated, focused and mouseover
: are not declared by the form developer. Instead, they are set for each item by forms viewer software based on system events.

bgcolor, fontcolor, labelbgcolor and labelfontcolor
: specify the colors for an item or its label using either predefined names or RGB triplets.

borderwidth and labelborderwidth
: control whether an item or its label has a 3D border.

coordinates
: receives the location of a mouse click on an image.

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.

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

filename and mimetype
: give additional information about an enclosed document.

fontinfo and labelfontinfo
: options define the typeface, point size, and special effects (bold, italics, underline) for the font used to display the item's value or label.

format
: option contains subelements that parameterize input validation for the item's value.

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.

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.

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

mimedata
: used to store large binary data blocks encoded in base 64.

next and previous
: link the item into the tab order of the page.

printsettings
: parameterizes the paper rendition of a form.

saveformat and transmitformat
: control how the form is written (UFDL, 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, etc.

signature, signdatagroups, signer, signformat, signgroups, signitemrefs, signitems, signoptionrefs and signoptions
: work together to provide a full-featured digital signature as defined in Section 1.1 Goal 9.

triggeritem
: is set in the form globals to identify which action, button, or cell was pressed last (and thus resulted in form submission).

type
: specifies whether the action, button, or cell item will perform a network operation, print, save, digitally sign, etc.

url
: provides the address for a page switch, or for a network link or submission.

value
: holds the primary text associated with the item.

visible
: determines whether the item should be shown to the user or made invisible.

The content of an option can take one of three formats: simple character data, a compute, or an array of subordinate XML elements. The content attribute identifies which type of content will appear.

```
[12] <!ATTLIST %options; content (simple|compute|array) "simple">
```

The default is *simple*, and the content attribute is not required if the content is *simple*, in which case the option must contain text with no child elements:

```
[13] <!ELEMENT %options; (#PCDATA)>
```

If the content expresses a compute, then the content must be present with the value "compute". The option may contain a *cval* element, which should contain simple character data for the current computed value of the option. If the cval element is absent, XFDL defines this to be equivalent to the presence of <cval></cval>. The option must contain a *compute* element. The compute element should contain a computational expression. 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 element. In essence, an application can treat the compute content as character data unless it must run the computes. See Section 3 for details on how the compute expression is represented.

[14] <!ELEMENT %options; (cval?, compute)>

[15] <!ELEMENT cval (#PCDATA)>

[16] <!ELEMENT compute (#PCDATA)>

The third case for an option's content is an array of subordinate elements. In this case, the content attribute is required and must be set equal to "array". The option must contain one or more array elements:

[17] <!ELEMENT %options; (%ae;+)>

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

```
<bgcolor content="array">

        <ae>255</ae>

        <ae>248</ae>

        <ae>220</ae>

</bgcolor>
```

The array element takes the same content attribute that option elements do, and its contents are controlled by the value of this attribute in the same way.

This recursive definition permits arbitrary depth for XFDL arrays. In XFDL, the form developer is permitted to give names other than *ae* to array elements. The default array element name is ae, but a name can be assigned even if XFDL does not define it. For example,

```
<bgcolor content="array">

        <red>255</red>

        <green>248</green>

        <blue>220</blue>

</bgcolor>
```

A number of the XFDL-defined options use array elements (such as bgcolor, itemlocation, and format). XFDL does not often assign names to the array elements, so the default tag name of *ae* is used. Since the form developer can assign names to array elements, the parameter entity reference to "%ae;" can only be partially defined as follows:

[18] <!ENTITY % ae "(copies | dialog | length | message | orientation | pages | printpages | range | template | ae)">

2.4 Base-64 Encoded Binary Data Objects

In XFDL, the mimedata option is used to store base-64 encoded binary data such as digital signatures, images, enclosed word processing or spreadsheet documents, etc. Base-64 encoding uses no characters that are illegal in character data, so mimedata content can be stored in a mimedata option element as simple character data. The only caveat is that since binary data tends to be long, XFDL processors are expected to "pretty print" the lines of base 64 using tabs, spaces and linefeeds such that the content appears to be indented with respect to the mimedata tags in text editors that wrap lines after 80 characters.

However, since XML preserves whitespace in element content, base-64 decoders for XFDL must be able to ignore an arbitrary amount of whitespace in the data.

2.5 Scope Identifiers (sid)

An XFDL scope identifier, or *sid*, uniquely identifies an element within the scope of its logical parent. An XFDL element may have a *sid* attribute which uniquely identifies the form within a system of forms in a large deployment. 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.

The lexical structure of a *sid* differs from the XML language rule Name, which used to define the lexical structure of attribute values of type ID. The dash, period, and colon are not permitted in a *sid* due to conflicts with their use as the subtraction symbol, relative scope membership operator, and ternary conditional operator (?:), respectively. The lexical structure of a *sid* is not designed as a replacement for the XML ID feature, which assigns a globally unique name to an element.

2.6 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 digital signatures and for saving or transmitting the form.

3. The XFDL Compute System

An XFDL compute can appear between <compute> and </compute>. This section defines the default infix notation for expressing computation expressions. As other appropriate XML languages are approved, they could be used in the content of a compute by defining a format attribute for the compute start tag. The default should be "infix" but the enumeration could be extended to include the names of supported formats. This version of XFDL only defines the default infix notation:

[19] <!ATTLIST compute format (infix) #FIXED "infix">

Most XFDL processors only need to preserve the compute as character data, but some applications must parse the text of computes, constructing a list of 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. Except for some minor modifications, the language rules in this section appear in Figure 2 of [1] as rules 6 to 19.

3.1 Ignoring 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 3.4), 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 is ignored.

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.

[28] Compute ::= Expr | Decision '?' Compute ':' Compute

The decision logic can apply logical-or (||), logical-and (&&), and logical negation (!) to the results of logical comparisons. The logical operators are left associative, and the comparators 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).

[20] Decision ::= Decision ‘||’ AndDecision | AndDecision
[21] AndDecision ::= AndDecision ‘**&&**’ NotDecision | NotDecision
[22] NotDecision ::= ‘**!**’ Comparison | Comparison
[23] Comparison ::= ‘**(**’ Decision ‘**)**’ | Expr (‘<’ | ‘>’ | ‘<=’ | ‘>=’ | ‘==’ | ‘**!=**’) Expr

Note that since Decision is capable of performing comparisons on the results of mathematical expressions, a Decision can ultimately start with an Expr. Therefore, a simple LR-type parser is required by XFDL computes.

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 Value (defined later).

[24] Expr ::= Expr ‘**+**’ Term | Expr ‘**-**’ Term | Expr ‘**+.**’ Term | Term
[25] Term ::= Term ‘*****’ NFactor | Term ‘/’ NFactor | Term ‘**%**’ NFactor | NFactor
[26] Nfactor ::= Factor | ‘**-**’ Factor
[27] Factor ::= Value ‘^’ NFactor | Value

3.3 Definition of Value

A value can be a compute in parentheses, which provides an override for the order of operations. A value can be a quoted string (Section 3.4). A value can be an XFDL reference to an element whose text data should be obtained when the compute is evaluated (Section 3.5). Finally, a value can be obtained as the result of a function call (Section 3.6).

[28] Value ::= ‘**(**’ Compute ‘**)**’ | *qstring* | UFDLReference | FunctionCall

3.4 Quoted Strings

The rules for recognizing a quoted string are quite difficult to express in BNF, but they are the usual rules that many high-level programming languages use to process quoted strings. The language rules for computes permit the recognition of a quoted string token using the italicized token name *qstring*.

An XFDL quoted string must be surrounded by double quotes. Whitespace before the open quote and after the close quote is ignored. Double quotes can be included by escaping them with a backslash (\). The escape sequences \n and \t result in a newline 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., \\). Finally, any byte value except 0 can be inserted into the quoted string content using \x followed by a two-digit hexadecimal number.

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 double quote is an open double quote, then the closing quote, whitespace, and open quote are discarded from the input stream.

3.5 XFDL References to Elements

Because each XFDL element's scope identifier (sid) is unique 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 obtain the character data of the value option in the item Field2 *on the same page*. If Field2 is on a separate page, say Page2, then a compute in Field1 can still access its value using Page2.Field2.value.

XFDL references can also grow arbitrarily in the opposite direction to describe unbounded array element depth. This is accomplished by introducing a second scoping operator, the square brackets, to describe depth below the option level. For example, given the following piece of XFDL for a format option:

```
<format content="array">
        <ae>dollar</ae>
        <range content="array">
                <ae content="compute">
                        <cval>35</cval>
                        <compute> Bill.value * "0.05" </compute>
                </ae>
                <ae content="compute">
                        <cval>700</cval>
                        <compute> Bill.value </compute>
                </ae>
        </range>
</format>
```

the reference format[0] yields 'dollar' and the reference format[range][1] yields '700'. 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. However, the numeric position can also be used, e.g.format[1][1] also yields '700'.

The above description covers static references. Dynamic references are a second important component of the UFDL referencing model. The left associative operator ->, 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 a UFDL 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 a UFDL listbox or popup. As is discussed in Section 6, the value option of a list or popup is equal to the item reference of the cell item for the selected cell. 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.

Finally, note that XFDL references support forward referencing. An XFDL reference can refer to any option or array element.

[29] UFDLReference ::= StaticRef | StaticRef **‘->’** DynamicRef
[30] StaticRef ::= ItemRef **‘.’** OptionRef | OptionRef
[31] ItemRef ::= ((sid **‘.’**)? sid **‘.’**)? sid
[32] DynamicRef ::= DynamicRef **‘->’** OptionRef | OptionRef
[33] OptionRef ::= sid (**‘[’** (Digit+ | sid) **‘]’**)*

3.6 Function Call Syntax

Function calls run code that may be external to the XFDL form definition. A set of predefined functions for doing standard mathematical operations, string manipulations, etc. is given in Section 7. The LibName allows functions to be grouped into separate namespaces, but predefined functions do not require a LibName.

[34] FunctionCall := (LibName **‘.’**)? FunctionName **‘(’** (Compute (**‘,’** Compute)*)? **‘)’**
[35] LibName ::= sid
[36] FunctionName ::= sid

3.7 Representing and Running XFDL Computes

This section presents a high-level algorithm describing how a XFDL Compute System must run the computes in a form. When a form starts, it must run all computes to provide content for the current value tags. This is accomplished by passing *nil* to RunXFDLComputes() as the change list. Each time an event, such as user input or an API call, causes a change to the simple data content or current value of an option or array element, RunXFDLComputes() is called with a change list containing only the element that changed.

```
Function: RunXFDLComputes
Input: F (all form elements), C (computes), E (list of changed elements)
Output: Z (set of all elements that changed due to running computes)
Z = empty list
Do {
    NewChangeList = empty list
    For I = 1 to n(C) do
        Pertinent = (E == nil) ? true : false
        For J = 1 to n(E)
            If $C_I$ contains $E_J$, then Pertinent = true
        If Pertinent
            $F_P$ = parent element containing $C_I$
            If current value of $F_P$ is not equal to eval($C_I$)
```

```
                NewChangeList += $F_P$
                Z += $F_P$
                cval($F_P$) = eval($C_I$)
        E = NewChangeList
 } while (E is not empty)
 return Z
```

If a string of simple data is assigned to an element via a public API call (e.g. as the result of user input or server-side processing), then the compute and its current value are destroyed.

The algorithm refers to the computes using one-based indexing (even though the computes may not be represented by an array in a given implementation). The symbol *n* denotes the number of computes in the form. For each compute C_I, the expression tree is checked to see whether it contains a static or dynamic reference to any element E_J in the change list. If so, then C_I is dependent on E_J and must be evaluated. The result of eval(C_I) does not equal the current content of the element parent F_P of C_I, then F_P is added to the new change list and the current value (the cval content) of F_P is set equal to the result of eval(C_I)—using a non-public API call such that the compute is not destroyed.

The algorithm does not show the semantics for dealing with circular references. Circular references are defined to be invalid XFDL. The computational output that results from running a circular chain of references is undefined. However, the behavioral result is defined. An XFDL processor should terminate in a finite amount of time upon encountering a chain of circular references. There is one exceptional case that the RunXFDLComputes algorithm is designed to permit. A compute that contains a self-reference is, in a graph theoretic sense, a circular reference. However, XFDL processors must support computes that use conditional logic to terminate computations after one iteration. Here is an example:

```
<user_email content="compute">

        <cval></cval>

        <compute>

                user_email == "" ? prefs.p1.ReturnAddress.value : user_email

        </compute>

</user_email>
```

In the first iteration, the current value of user_email is empty, so the compute runs and changes the current value to be equal to the content of a particular value option in another form called prefs. The change causes user_email to enter the NewChangeList. During the following iteration of the loop, the compute runs again, but the current value of user_email does not change, so user_email does not enter the new change list.

The evaluation function must perform run-time type identification on operands. The only permitted operation on strings is addition (by + or +.). Dates can only be added and subtracted. Numeric addition should only be performed if both operands are numeric.

4. Small XFDL Form Examples

The first example in Figure 1 is designed to show a whole XFDL form. After the XML prolog, the root XFDL element declares a version of 4.1.0. 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. This is not needed if the color is given by name, but it is required if the background color is given as an RGB triplet. XFDL options and array elements are consistent in their use of element depth.

The page global options also contain a label option that declares the caption bar text for the window used to display the form page. Note that 'label' is one of those keywords 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. The XFDL parser distinguishes a global option from an item based on the absence or presence, respectively, of the 'sid' attribute.

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 1: A Simple XFDL Form

```
<?xml version="1.0"?>

<XFDL version="4.1.0">

       <bgcolor content="array">

        <ae>128</ae> <ae>128</ae> <ae>128</ae>

       </bgcolor>

       <page sid="Pythagorean_Theorem">

        <bgcolor content="array">

                <ae>192</ae> <ae>192</ae><ae>192</ae>

        </bgcolor>

        <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 content="compute">
                            <cval>5</cval>
                            <compute>sqrt(A.value^"2" + B.value^"2")</compute>
                    </value>
            </field>
           </page>
    </XFDL>
```

The second example in Figure 2 does not include the XML prolog nor 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 1.

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 6), the input type is not named and would therefore appear between the <ae> and </ae> tags. However, the form developer can assign names to array elements that are not required by the XFDL specification to have a name. The second and third array elements in the format option are 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 prepended to the user's input.

The last array element declared in the format option in Figure 2 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. The XFDL offers what is known as a fine-grain compute system.

The second item element in Figure 2 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.

Figure 2: Example of Suboption Array Elements

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

5. 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


</item>
```

Notes:

i) 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

custom items outlined in this specification.

ii) The sid attribute is mandatory.

iii) The value of each item sid must be unique in the page.

The *sid* attribute uniquely identifies an item. 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 allowed.

See the section "6. Details on Options and Array Elements" for descriptions of each option type.

5.1 action

Specifies form-initiated actions that execute automatically. The actions can be any of the following types: link, replace, submit, done, display, print, cancel.

See the *type* description in the 'Options' section for a description of each of these actions.

Action items can be defined to occur only once or repeat at specified time intervals. They can be defined to occur 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. In fact, if the form contains only hidden items such as *action* items, then the whole form operates in the background. Such forms are called daemon forms.

5.1.1 Available Options

activated, active, data, datagroup, delay, transmitformat, type, url

5.1.2 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.

5.1.3 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 content="array">

                <ae>repeat</ae>

                <ae>600</ae>

        </delay>

        <type>submit</type>

        <url content="array">

                <ae>http://www.server.com/cgi-bin/recv_status</ae>

        </url>

</action>
```

5.2 box

Specifies a square 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.

5.2.1 Available Options

bgcolor, borderwidth, fontinfo, itemlocation, size

5.2.2 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).

5.2.3 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 content="array">

                <ae>blue</ae>

        </bgcolor>

        <size content="array">

                <ae>25</ae>

                <ae>4</ae>

        </size>

</box>
```

5.3 button

Specifies a click button that performs an action when selected. Buttons can request data from a web server, submit or cancel the form, sign the form, save it to disk, or enclose external files.

5.3.1 Available Options

activated, active, bgcolor, borderwidth, coordinates, data, datagroup, focused, fontcolor, fontinfo, format, help, image, itemlocation, justify, mouseover, next, signature, signdatagroups, signer, signformat, signgroups, signitemrefs, signitems, signoptionrefs, signoptions, size, transmitformat, type, url, value

5.3.2 Usage Notes

1. The button’s label is defined by the *value* option. If no *value* option exists, the default label is blank.

2. When setting the *size* option of a button, the height and width of the button will be 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 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.

4. The *format* option is available in buttons in order to force users to sign forms before submitting them.

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.)

Behavior of Buttons in Digital Signatures

1. A digital signature button is the means by which the user can digitally 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.

   Important: At a minimum, the *triggeritem* and *coordinates* options should always 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 (like *odbc_rowcount*), 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 digital signature, however, then the formulas will become unfrozen, and will change dynamically as normal.

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

5.3.3 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 content="array">
                <ae>Helvetica</ae>
                <ae>18</ae>
                <ae>bold</ae>
                <ae>italic</ae>
        </fontinfo>
        <type>done</type>
        <url content="array">
                <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 content="array">
                <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.

5.4 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 cancelling, 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.

5.4.1 Available Options

activated, active, data, datagroup, group, label, transmitformat, type, url, value

5.4.2 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 content="array">
                <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>
```

5.4.3 Usage Notes

1. Use the *type* option to establish a cell's behavior. Select cells that 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. 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:

   ```
   <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.

5.5 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.

5.5.1 Available Options

active, bgcolor, editstate, focused, fontcolor, fontinfo, help, itemlocation, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, size, value

5.5.2 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).

5.5.3 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 content="array">

                <ae>Times</ae>

                <ae>14</ae>
                <ae>bold</ae>
        </labelfontinfo>
        <labelfontcolor content="array">
                <ae>blue</ae>
        </labelfontcolor>
</check>
```

5.6 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.

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

5.6.1 Available Options

activated, active, bgcolor, editstate, focused, fontcolor, fontinfo, format, group, help, itemlocation, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, previous, size, value

5.6.2 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. Cells that have a *label* option will display that label in the list. Otherwise, the *value* of the cell will be displayed. When a cell is selected, the *value* of that cell will be displayed in the combobox and stored internally.

3. 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.

4. Combobox, popup, and list items with the same group reference display the same group of cells.

5. When first viewed, a combobox will display its *value*. If no value is set, the combobox will be empty.

6. The *value* option will contain one of the following:

    - The *value* of the most recently chosen selection.

    - Nothing if an action was most recently chosen.

    - The text entered if something was typed in most recently.

7. 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).

8. The *label* option sets the text displayed above the item, as with a field.

9. 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.

10. 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.

11. If any two comboboxes, lists, or popups use the same set of cells, they must apply the same formatting.

5.6.3 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>
```

5.7 data

Stores an information object such as an image, a sound, or an enclosed file in an XFDL form. 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**.

5.7.1 Available Options

datagroup, filename, mimedata, mimetype

5.7.2 Usage Notes

1. Store the data in the *mimedata* option, and the data's MIME type in the *mimetype* option.

2. 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.

5.7.3 Example

This is an example of a *data* item produced as the result of enclosing a file (the data component used here is artificial, and is only for demonstration purposes).

```
<data sid="Supporting_Documents_1">

        <filename>smithltr.doc</filename>       <datagroup content="array">

                <ae>Supporting_Documents</ae>

        </datagroup>

        <mimetype>application/uwi_bin</mimetype>

        <mimedata>R0lGODdhYABPAPAAAP///wAAACwAAAAAYABPAAAC/4SPqcvtD02Y

Art68+Y7im7ku2KkzXnOzh9v7qNw+k+TbDoLFTvCSPzMrS2YzmTE+p

yai3YUk9R6hee2JFP2stju+uG0ptvdeKptb+cX8wfY1jdYU4ehKDi3pdJw

44yAJEqcW28cA5M0oEKnqKasZwydrK9Wo6JTtLG9p5iwtWi8Tbi/b7E0

rvKixzbHJyrDq2uNggaXUs1NlLi36AW3AGv7VWhIPA7TzvdOGi/vvr0Of

ft3Nrx89JewCQJYTirxi2PwgnRpNoMV5FIIboOnqTszFLFIMhQVI0yOz

        </mimedata>

</data>
```

5.8 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.

5.8.1 Available Options

active, bgcolor, editstate, focused, fontcolor, fontinfo, format, help, itemlocation, justify, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, size, value

5.8.2 Usage Notes

1. When setting the *size* option of a field, the height and width of the field will be based on the average character size for the font in use (set with the *fontinfo* option).

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.

5.8.3 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 content="array">

                <ae>20</ae>

                <ae>1</ae>

        </size>

        <fontinfo content="array">

                <ae>Courier</ae>

                <ae>12</ae>

                <ae>plain</ae>

        </fontinfo>

        <labelfontinfo content="array">

                <ae>Helvetica</ae>

                <ae>12</ae>

                <ae>plain</ae>

        </labelfontinfo>
        <labelfontcolor="array">
                <ae>blue</ae>
        </labelfontcolor>
</field>
```

5.9 help

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

5.9.1 Available Options

active, value

5.9.2 Usage Notes

The *help* item's *value* option contains the help message text.

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.

5.9.3 Example

This is an example of a button for which help information is available.

The following is the button definition with the *help* item's item reference in the *help* option:

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

```
</button>
```

The following example shows the *help* item referred to in the button definition. The contents of the *value* option are used 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 do

</help>
```

5.10 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.

5.10.1 Available Options

active, bgcolor, fontcolor, fontinfo, format, help, image, itemlocation, justify, size, value

5.10.2 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 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.

5. The label's background color defaults to being transparent– and thus the label will take the background color of whatever item it is over. For example, it is possible to place a label inside a colored box (in order to make a title section that stands out) without specifying a background color for the label.

5.10.3 Example

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

```
<!--Specify right justification for this 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 content="array">

                <ae>Times</ae>

                <ae>16</ae>

                <ae>italic</ae>

        </fontinfo>

</label>
```

5.11 line

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

5.11.1 Available Options

fontcolor, fontinfo, itemlocation, size, thickness

5.11.2 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.

5.11.3 Example

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

```
<line sid="BLUE_LINE">

        <size content="array">

        <ae>40</ae>
```

```
                <ae>0</ae>
            </size>
            <thickness>5</thickness>
    </line>
```

5.12 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.

5.12.1 Available Options

active, bgcolor, editstate, focused, fontcolor, fontinfo, format, help, itemlocation, label, labelbgcolor, labelborderwidth, labelfontcolor, labelfontinfo, mouseover, next, size, value

5.12.2 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. Cells that have a *label* option will display that label in the list. Otherwise, the *value* option of the cell will be displayed.

3. 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>
   ```

4. 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. *List*, *combobox* and *popup* items with the same group reference display the same group of cells.

6. 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".

7. Define the list’s label using the *label* option.

8. 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).

9. 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.

10. 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.

11. If any two comboboxes, lists, or popups use the same set of cells, they must apply the same formatting.

5.12.3 Example

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

Here is the list definition.

```
<list sid="MAINMENU_LIST">

        <group>list_Group</group>

        <label>Options Menu</label>

        <labelfontcolor content="array">

                <ae>blue</ae>

        </labelfontcolor>

        <size content="array">

                <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 content="array">
```

```
                <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>
```

5.13 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.

5.13.1 Available Options

activated, active, bgcolor, borderwidth, editstate, focused, fontcolor, fontinfo, group, help, itemlocation, justify, label, mouseover, next, size, value

5.13.2 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. Cells that have a *label* option will display that label in the list. Otherwise, the *value* of the cell will be displayed. When a cell is selected, the *value* of that cell will be displayed in the popup.

   For example, if cell had a *value* of "USA", and a *label* of "United States of America", the full version would be shown in the popup list. Once the cell was selected, the popup would display the abbreviation.

3. 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.

4. *Popup, combobox* and *list* items with the same group reference display the same group of cells.

5. 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".

6. 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).

7. The *label* option contains the popup’s default label. When the *value* option is empty, the default label displays. Otherwise, the label of the cell identified in the *value* option appears.

8. 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.

9. If any two comboboxes, lists, or popups use the same set of cells, they must apply the same formatting.

5.13.3 Example

This is an example of a popup 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>
```

5.14 radio

Intended for use with one or more other *radio* button 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.

5.14.1 Available Options

active, bgcolor, borderwidth, editstate, focused, fontcolor, fontinfo, group, help, itemlocation, label, mouseover, next, size, value

5.14.2 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).

5.14.3 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*.

5.15 signature

Contains a digital 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 digital 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.

5.15.1 Available Options

mimedata, signature, signdatagroups, signer, signformat, signgroups, signitemrefs, signitems, signoptionrefs, signoptions

5.15.2 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*, *signoptionrefs*, and *signoptions* options are copied from the button description to the signature description.

3. 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.

4. 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.

5. 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.

6. 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.

7. To use digital signatures, it is necessary for the user to obtain a digital signature certificate.

5.15.3 Example

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

```
<button sid="empSigButton">

        <type>signature</type>

        <value content="compute">

                <compute>signer</compute>

        </value>

        <signer></signer>

        <format content="array">

                <ae>string</ae>

                <ae>mandatory</ae>

        </format>

        <signformat>application/x-xfdl;csp="Microsoft Base Cryptographic Provi

        <signoptions content="array">

                <ae>omit</ae>

                <ae>triggeritem</ae>

                <ae>coordinates</ae>

        </signoptions>

        <signitemrefs content="array">

                <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/x-xfdl;csp="Microsoft Base Cryptographic

Provider v1.0";csptype=rsa_full;hashalg=sha1</signformat>

        <signer>Jane D Smith, jsmith@insurance.com</signer>

        <signature>PAGE1.empSignature</signature>

        <signitemrefs content="array">

                <ae>omit</ae>

                <ae>PAGE1.mgrSigButton</ae>

                <ae>PAGE1.admSigButton<ae>

                <ae>PAGE1.empSignature</ae>

                <ae>PAGE1.mgrSignature</ae>

                <ae>PAGE1.admSignature</ae>

        </signitemrefs>

        <signoptions content="array">

                <ae>omit</ae>

                <ae>triggeritem</ae>

                <ae>coordinates</ae>

        </signoptions>

        <mimedata>MIIFMgYJKoZIhvcNAQcCoIIFIzCCBR8CAQExDzANBgkg

        AQUFADALB\ngkqhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl

        6hMrc5DySSP+X5j\nANfBGSOI\n9w0BAQQwDwYDVQQHEwhJbn<

        Rlcm5ldDEXMBUGA1UEChM\nOVmVyaVNpZ24sIEluYy4xNDAKn

        1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmRdWFsIFN1YnNjcmliy

        ZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCARExETA

        </mimedata>

</signature>
```

5.16 spacer[Display]

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

5.16.1 Available Options

fontinfo, itemlocation, label, size

5.16.2 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).

5.16.3 Example

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

```
<spacer sid="3_SPACER">

        <size content="array">

                <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>
```

5.17 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. Typically, items placed in the toolbar are those users are to see 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.

5.17.1 Available Options

bgcolor, mouseover

5.17.2 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.

5.17.3 Example

This example shows a toolbar that contains a label, a spacer, and two buttons.

Here is the toolbar definition:

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

Here is an item that will appear in the toolbar.

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

5.18 <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.

5.18.1 Available Options

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

5.18.2 Usage Notes

The naming conventions for a custom item are as follows:

- It must begin with an alphabetic character.
- It must contain characters only from the following list: A-Z, a-z, 0-9 and underscore.
- It must contain an underscore.

5.18.3 Example

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

```
<ma_event sid="STATUS_EVENT">

        <active>off</active>

        <ma_id>UF45567  /home/users/preferences01</ma_id>

</ma_event>
```

6. Details on Options and Array Elements

Syntax

For simple character data content:

```
<optionTag>content</optionTag>
```

For computed options:

```
<optionTag content="compute">

        <cval>current value data</cval>

        <compute>expression</compute>

</optionTag>
```

For array options:

```
<optionTag content="array">

        <!suboption elements -->

</optionTag>
```

An option defines a characteristic given to a form, a page, or an item. For example, a fontinfo option set at the form global level defines the default font characteristics for the entire form. A fontinfo option set at the item level defines the font characteristics for only the item it is in.

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. The content attribute identifies which type of content will appear.

Simple Character Data

The default is simple, and the content attribute is not required if the content is simple, in which case the option must contain text with no child elements. For example:

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

Compute

If the content expresses a compute, then the content attribute must be present and be set equal to "compute". The option may contain a cval element, which should contain simple character data for the current computed value of the option. If the cval element is absent, XFDL defines this to be equivalent to the presence of <cval></cval>. The option must contain a compute element. The compute element should contain a computational expression. For example:

```
<value content="compute">

        <cval></cval>

        <compute>

                price1Field.value+price2Field.value*"0.07"

        </compute>

</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 element. In essence, an application can treat the compute content as character data unless it must run the computes. See Section 3 for details on how the compute expression is represented.

Array

The third case for an option's content is an array of subordinate elements. In this case, the content attribute is required and must be set equal to "array". The option must contain one or more array elements. For example:

```
<bgcolor content="array">
```

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

The array element takes the same content attribute that option elements do, and its contents are controlled by the value of this attribute in the same way. This recursive definition permits arbitrary depth for XFDL arrays.

Array Element Names

In XFDL, the form developer is permitted to give names other than ae to array elements. The default array element name is ae, but a name can be assigned even if XFDL does not define it. For example,

```
<bgcolor content="array">
        <red>255</red>
        <green>248</green>
        <blue>220</blue>
</bgcolor>
```

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.

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">
        <fontinfo content="array">
                <ae>Helvetica</ae>
                <ae>12</ae>
                <ae>plain</ae>
        </fontinfo>
```

Defining Form Global and Page Global Options

Form global options are optional and must be defined after the XFDL start tag, but before the first page in a form. Page global options are optional and must be defined after the page declaration, but before the first item in a page. To determine whether an option is a valid form global or page global option, see that option's description later in this specification.

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 ar RGB triplet is: <bgcolor content="array"> <ae>*red*</ae> <ae>*green*</ae> <ae>*blue*</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
long int	whole number in the range 0 to 2,147,483,647
short int	whole number in the range 0 to 255
unsigned	whole number in the range 0 to 65,535

6.1 activated

Specifies whether an item, page, or form is currently activated by the user or not. This option is set by code outside XFDL.

6.1.1 Syntax

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

Expression	**Setting**	**Description**
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

6.1.2 Available In

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

6.1.3 Example

The following example shows a button that changes color based on whether it is currently activated.

```
<button sid="saveButton">

        <type>save</type>

        <value>Save</value>

        <activated>off</activated>

        <bgcolor content="array">

                <ae content="compute">

                        <compute>activated=="on" ? "255" : "100"</compute>

                </ae>

                <ae content="compute">

                        <compute>activated=="on" ? "255" : "100"</compute>

                </ae>

                <ae content="compute">

                        <compute>activated=="on" ? "255" : "100"</compute>

                </ae>

        </bgcolor>

</button>
```

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

6.1.4 Usage Notes

1. Default: **off**

2. Any pre-defined setting for *activated*, including a formula setting, will be destroyed as soon as the first activated event appears. The *activated* option is not intended to be set by XFDL script, but rather by external forces—a form viewing program, for example.

3. *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.

4. The *activated* option is not included in form descriptions that are saved or transmitted.

5. 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 displayed on screen, and off when it is not.
- form – a form sets *activated* to on while it is displayed on screen, and off when it is not.

6.2 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.

6.2.1 Syntax

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

Expression	Setting	Description
status	**on**	item is active
	off	item is inactive

6.2.2 Available In

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

6.2.3 Example

This sample specifies the item is active.

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

6.2.4 Usage Notes

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

6.3 bgcolor[Display]

Defines the background color of a page or an item.

6.3.1 Syntax

```
<bgcolor content="array">

        RGB triplet

</bgcolor>


or


<bgcolor content="array">

        <ae>color_name<ae>

</bgcolor>


Note: Either format is acceptable.
```

Expression	***Setting***	**Description**
color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See ‘Data Type Designators' for the triplet.

6.3.2 Available In

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

6.3.3 Examples

These samples both set the background color to forest green.

```
<bgcolor content="array">

        <ae>forest green</ae>

</bgcolor>


<bgcolor content="array">

        <ae>34</ae>

        <ae>139</ae>

        <ae>34</ae>

</bgcolor>
```

6.3.4 Usage Notes

1. The **transparent** color has no RGB equivalent.

2. ***Default:*** varies depending on the object
   Form: **white**
   Page: the form background color
   Item: depends on the item type—
   button items: **gray** (or **grey**)
   check, combobox field, list, popup, and radio items: **white**
   label items: **transparent** (version 4.0.1 and greater)
   all other items: the background color of the page

6.4 borderwidth[Display]

Defines the width of an item’s border. The unit of measurement is pixels.

6.4.1 Syntax

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

Expression	Setting	Description
width	*short int*	the width of the border in pixels

6.4.2 Available In

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

6.4.3 Example

This sample sets the border width to five pixels.

```
<borderwidth>5</borderwidth>
```

6.4.4 Usage Notes

Default: varies depending on the item type

- *box* and *label* items: zero pixels
- all other visible items: one pixel

6.5 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 top, left corner. The coordinates map the intersection closest to the mouse pointer's position.

6.5.1 Syntax

```
<coordinates content="array">

        <ae>X_coordinate</ae>

        <ae>Y_coordinate</ae>

</coordinates>
```

Expression	**Setting**	**Description**
X_coordinate	*coordinate*	the coordinate on the X axis
Y_coordinate	*coordinate*	the coordinate on the Y axis

6.5.2 Available In

button

6.5.3 Example

When a user clicks on a button containing an image, a *coordinates* option statement is inserted into the *button* 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 content="array">

        <ae>180</ae>

        <ae>255</ae>

</coordinates>
```

6.5.4 Usage Notes

Default: none

6.6 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**.

6.6.1 Syntax

```
<data>data_item</data>
```

Expression	Setting	Description
data_item	*string*	the item tag of the data item to associate with the action button, or cell

6.6.2 Available In

action, button, cell

6.6.3 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.)

6.6.4 Usage Notes

1. A *data* option specifies only zero or one data items.

2. 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.

3. 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.

4. 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.

6.7 datagroup

Identifies a group or folder of enclosed files. Each enclosed file can belong to several datagroups, and each datagroup can contain several enclosed files.

6.7.1 Syntax

```
<datagroup content="array">


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


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 access
```

Expression	Setting	Description
datagroup reference	*string*	identifies a datagroup

6.7.2 Available In

action, button, cell, data

6.7.3 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 a *action*, *button,* or *cell* item, it would mean the user could store the enclosure in one of the three datagroups.

```
<datagroup content="array">

        <ae>Business_Letters</ae>

        <ae>Personal_Letters</ae>

        <ae>Form_Letters</ae>

</datagroup>
```

6.7.4 Usage Notes

1. ***Default***: none

2. 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.

3. Items that handle enclosed files perform *enclose*, *extract*, *remove*, and *display* actions. These actions types are set using the *type* option.

4. 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.

5. 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.

6.8 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.

6.8.1 Syntax

```
<delay content="array">

        <ae>repeat factor</ae>

        <ae>interval</ae>

</delay>
```

Expression	Setting	Description
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 occurrence actions. The unit of measurement is seconds.
	-1	perform the action before the page displays. Only val factor of *once*.

6.8.2 Available In

action

6.8.3 Example

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

```
<delay content="array">

        <ae>once</ae>

        <ae>900</ae>

</delay>
```

6.8.4 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.

6.9 editstate

Defines one of three possible edit states for modifiable items.

6.9.1 Syntax

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

Expression	Setting	Description
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

6.9.2 Available In

check, field, list, popup, radio

6.9.3 Example

This sample sets the editstate to readonly.

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

6.9.4 Usage Notes

1. **Default:** *readwrite*.

2. The *writeonly* setting applies only to fields. 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.

6.10 filename

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

6.10.1 Syntax

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

Expression	Setting	Description
name of file	*string*	the name of the enclosed file

6.10.2 Available In

data

6.10.3 Example

This sample specifies the name of an enclosed file.

```
<filename>std_logo.frm</filename>
```

6.10.4 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 *.frm* extension.

6.11 focused

Specifies whether an item, page, or form currently has the input focus. This option is set by code outside XFDL.

6.11.1 Syntax

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

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

6.11.2 Available In

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

6.11.3 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 content="array">

                <ae content="compute">

                        <compute>activated=="on" ? "255" : "0"</compute>

                </ae>

                <ae content="compute">

                        <compute>activated=="on" ? "255" : "0"</compute>

                </ae>

                <ae content="compute">

                        <compute>activated=="on" ? "255" : "255"</compute>

                </ae>

        </bgcolor>

</button>
```

6.11.4 Usage Notes

1. Default: **off**

2. Any pre-defined setting for *focused*, including a formula setting, will be destroyed as soon as the first activated event appears. The *focused* option is not intended to be set by XFDL script, but rather by external forces—a form viewing program, for example.

3. *focused* is set to **on** when an item, page, or form receives the input focus, and is set to **off** when it does not.

4. An object's *focus* 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 *focus* set to **on**, even if the page is minimized or is not the currently active application on the desktop.

5. 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.

6. When a form viewing application is closing a form, it should set all *focus* options to **off** and then resolve all formulas before shutting down.

7. *focused* may only be set by a desktop form viewing application.

8. The *focused* option is not included in form descriptions that are saved or transmitted.

6.12 fontcolor[Display]

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.

6.12.1 Syntax

```
<fontcolor content="array">

        <ae>color name</ae>

</fontcolor>


<fontcolor content="array">

        RGB triplet

</fontcolor>


Note: Either format is acceptable.
```

Expression	Setting	Description
color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See ‘Data Type Designators‘ for the sy of an RGB triplet.

6.12.2 Available In

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

6.12.3 Examples

These samples both set the background color to chocolate.

```
<fontcolor content="array">
```

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

<fontcolor content="array">
        <ae>210</ae>
        <ae>105</ae>
        <ae>30</ae>
</fontcolor>
```

6.12.4 Usage Notes

Default: black

6.13 fontinfo[Display]

Defines the font name, point size, and font characteristics for the text portion of an item.

Note: The font selected for an item influences the item’s size.

6.13.1 Syntax

```
<fontinfo content="array">
        <ae>font name</ae>
        <ae>point size</ae>
        <ae>weight</ae>
        <ae>effects</ae>
        <ae>form</ae>
</fontinfo>


Note: weight, effects, and form are optional.
```

Expression	**Setting**	**Description**
font name	*string*	the name of the font
point size	*short int*	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

6.13.2 Available In

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

6.13.3 Example

This sample sets the font information to Times 14, bold italic.

```
<fontinfo content="array">

        <ae>Times</ae>

        <ae>14</ae>

        <ae>bold</ae>

        <ae>italic</ae>

</fontinfo>
```

6.13.4 Usage Notes

1. ***Defaults:***

    - font name: Helvetica
    - point size: 8
    - weight: plain
    - effects: not underlined
    - form: not italics

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 and font sizes:
   Fonts: Courier, Times, Symbol (symbol), Helvetica, and Palatino
   Sizes: 8, 9, 10, 11, 12, 14, 16, 18, 24, 36, 48

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.

6.14 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).

6.14.1 Syntax

```
<format content="array">
        <ae>data type</ae>
        <ae>format flag</ae>
        <ae>check flag</ae>
</format>


Notes:
i) Multiple flags are valid.
ii) data type is mandatory and must appear first; the flags are optional
and may appear in any order.
```

Expression	Setting	Description
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 inp

6.14.2 Available In

button, combobox, field, label, list, popup

6.14.3 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 content="array">
        <ae>integer</ae>
        <ae>comma_delimit</ae>
        <range content="array">
                <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 content="array">

        <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., 15 June 1999).

```
<format content="array">

        <ae>date</ae>

        <ae>long</ae>

</format>
```

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

```
<format content="array">

        <ae>string</ae>

        <template content="array">

                <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 content="array">

        <ae>string</ae>

        <ae content="compute">

                <compute>page1.allowIncompleteCheck=="on"?"optional" : "mandat

        </ae>

</format>
```

6.14.4 Data Types[Types]

XFDL supports the following data types:

Data Type	**Description**	**Format**
<ae>string</ae>	free form character data up to 32K long	Any cha
<ae>integer</ae>	a positive or negative whole number in the range of -2,147,483,648 to +2,147,483,647	Any wh
<ae>float</ae>	a positive or negative floating point decimal number in the range of 1.7 * 10-308 to 1.7 * 10308	Any dec
<ae>dollar</ae>	a fixed point decimal number with a scale of 2 and a range equal to the range of a **float**	Any nur Automa end, if n specifie
<ae>date</ae>	a date including *day-of-month*, *month*, and *year*	This for 3 Mar 9
<ae>day_of_week</ae>	the name or number of a day of the week	This for
<ae>month</ae>	the name or number of a month	This for
<ae>day_of_month</ae>	the number of a day of the month	Number
<ae>year</ae>	a numeric year designation	This for 1996 | 2
<ae>time</ae>	a time value containing hours and minutes from the 12 hour or the 24 hour clock	This for 11:23 P.
<ae>void</ae>	disable entire *format* option (including data type, checks, and formats)	No effec field

6.14.5 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 parent
<ae>add_ds</ae>	Add a dollar sign to the start of the value (*dollar* fields only).
<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 le for titles and proper names.
<ae>short</ae>	Display dates and times using the following formats:

	• day_of_week - numeric value in range 1 to 7 where 1 r • 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 ('9 era designator is B.C. ('98 B.C.) • date - year as four digits, month as two digits, and day- digits, organized in YMD order; no punctuation (1998- • time - 24 hour clock (as in 23:30)
<ae>long</ae>	Display dates and times using the following formats: • day_of_week - name in full as in Monday • day_of_month - two digits plus suffix as in 1st • month - name in full as in January • year - four-digit numeric format, 'before Christ' era de (2000 B.C.) • date - long year, long month, and long day-of-month fo DMY order; no punctuation (29th April 1998) • time - 12 hour clock with the time of day suffix (A.M. P.M.)
<ae>numeric</ae>	Display dates and times using numeric values and, possibly, t • day_of_week - 2 digits in range 01 to 07 where 01 repr • 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 s • date - month and day-of-month formats above, - year fo 'before Christ' era designator is minus sign - organized in YMD order; no punctuation - Examples: 19980429, -19980429 • time - 24 hour clock (as in 23.30)
<presentation>yy/mm/dd </presentation>	Available only when formatting dates, to create custom templ of dates, using Y for year, M for month, and D for day Example: "date", presentation= "YY/MM/DD" -this could yield 98/12/23
<ae>void</ae>	No formatting is applied

6.14.6 Check Flag

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.

Important: 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 use
<range content="array"> <ae>*low*</ae> <ae>*high*</ae> </range>	The field's value must be in the range specified. The range n numeric, days of the week, days of the month, or months. Ranges cannot vary from high to low. For example, 10 to one 1900, etc. are invalid.
<length content="array"> <ae>*min*</ae> <ae>*max*</ae> </length>	Restrict the length of the formatted input data to a minimum maximum of "max" bytes.
<template content="array"> <ae>*a*</ae> <ae>*b*</ae> <ae>...</ae> </template>	This is a list of formats permitted for the field. There is no re number of formats. Field contents must match one of the forr 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 (wh
<message>help</message>	Sets the error message that is displayed if the user input fails The default message is, "This entry is invalid. Please try agai
<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.

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

Data Type	**Applicable Format Flags**	**Applicable Check F**
string	lower_case, upper_case, title_case	case_insensitive, fail_ ignore_checks, length optional, range, temp
integer	bracket_negative, comma_delimit, space_delimit	fail_checks, ignore_c mandatory, optional,

float	bracket_negative, comma_delimit, space_delimit	fail_checks, ignore_c mandatory, optional,
dollar	add_ds, bracket_negative, comma_delimit, space_delimit	fail_checks, ignore_c mandatory, optional,
date **year** **month** **day_of_month** **day_of_week** **time**	long, short, numeric	case_insensitive, fail_ ignore_checks, length optional, range, temp
void	No formatting or type checking is done	No checking or type

6.14.8 Usage Notes

1. If a *format* flag conflicts with the data type, the *format* flag will be ignored.

2. 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.

3. **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 short if the input is between 0:00 and 12:59, and to long otherwise.

4. **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.

5. 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.
6. When applying a *format* to a combobox, list, or popup, 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.
7. If any two comboboxes, lists, or popups use the same set of cells, they must apply the same formatting.
8. 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.
9. The *void* format flag facilitates the modification of a format statement by a formula. It is ignored by the formatting system.
10. For details on using the *format* option in buttons, see the Usage Notes in the button item description.

6.15 group

Provides a method of grouping items together. Items with the same *group reference* are considered members of the same group. Examples of grouped items are radio buttons and cells.

6.15.1 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
```

Expression	Setting	Description
group reference	*string*	identifies the group

6.15.2 Available In

cell, combobox, list, popup, radio

6.15.3 Example

This sample associates the item with the group *coverage_type*.

```
<group>coverage_type</group>
```

6.15.4 Usage Notes

1. **Default**: none
2. List and popup items are populated with cells that have the same group reference as the item. It is possible to have several 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.
3. All radio items having the same group reference will form a mutually exclusive group.

6.16 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.

6.16.1 Syntax

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

Expression	Setting	Description
item reference	*string*	identifies the *help* item

6.16.2 Available In

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

6.16.3 Example

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

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

6.17 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.

6.17.1 Syntax

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

Expression	Setting	Description
item reference	*string*	identifies the *data* item

6.17.2 Available In

button, label

6.17.3 Example

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

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

6.17.4 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.

6.18 itemlocation[Display]

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 might dynamically alter the size of an item.

Each specification 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: an absolute positioning scheme and a relational positioning scheme. The absolute positioning scheme anchors the top left corner of an item to a particular pixel on the displayed page, whereas the relational positioning scheme places items on the page in relation to one another. The two schemes can be combined to gain the advantages of both.

For more information on the two schemes, see Absolute Positioning Scheme and Relational Positioning Scheme.

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.

6.18.1 Syntax:

```
<itemlocation content="array">
        <ae content="array">
                specification₁
        </ae>
...
        <ae content="array">
                specificationₙ
        </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 tag₁,
     item tag₂



Notes:
i) There is no restriction on the number of specifications.
ii) x-coordinate and y-coordinate are negative only when the modifier is offs
```

```
iii) <item tag2> is valid only with the modifiers alignhorizbetween and

alignvertbetween.
```

Expression	Setting	Description
modifier	*(see below)*	the type of modification to apply to the item's locatio
x-coordinate	*short (must be positive if modifier is absolute)*	• the horizontal distance in pixels from the form's top the modifier *absolute*); or • the horizontal distance in pixels from the item's top original position to its new offset position (with the n
y-coordinate	*short (must be positive if modifier is absolute)*	• the vertical distance in pixels from the form's top le modifier *absolute*); or • the vertical distance in pixels from the item's top lef original position and to its new offset position (with t
item tag	*string*	identifies the reference point item

6.18.2 Modifiers

There are four types of modifiers:

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

Position Modifiers

For the Absolute Positioning Scheme:

Modifier	Description
absolute	Place top left corner of item on the pixel noted in the x-coordinate and y settings.
offset	Place item so that it is offset from its original location by the measurem 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:

Note: A specification containing the *within* modifier must be the first specification in the list.

Modifier	**Description**
above	Place item a small distance above reference point item; align left edges.
after	Place item a small distance after reference point item; align top edges.
before	Place item a small distance before reference point item; align top edges.
below	Place item a small distance below reference point item; align left edges.
within	Assign item to the toolbar.

Alignment Modifiers (Relational Positioning only)

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

Modifier	**Description**
alignb2b	Align bottom edge of item with bottom edge of reference point iten
alignb2c	Align bottom edge of item with vertical center of reference point it
alignb2t	Align bottom edge of item with top edge of reference point item.
alignc2b	Align vertical center of item with bottom edge of reference point it
alignc2l	Align horizontal center of item with left edge of reference point ite
alignc2r	Align horizontal center of item with right edge of reference point it
alignc2t	Align vertical center of item with top edge of reference point item.
alignhorizbetween	Align horizontal center of item between right edge of first reference left edge of second reference point item.
alignhorizc2c	Align horizontal center of item with horizontal center of reference p below.
alignl2c	Align left edge of item with horizontal center of reference point ite
alignl2l	Align left edge of item with left edge of reference point item.
alignl2r	Align left edge of item with right edge of reference point item.
alignr2c	Align right edge of item with horizontal center of reference point it
alignr2l	Align right edge of item with left edge of reference point item.
alignr2r	Align right edge of item with right edge of reference point item.
alignt2b	Align top edge of item with bottom edge of reference point item.
alignt2c	Align top edge of item with vertical center of reference point item.
alignt2t	Align top edge of item with top edge of reference point item.
alignvertbetween	Align vertical center of item between bottom edge of first reference edge of second reference point item.
alignvertc2c	Align vertical center of item with vertical center of reference point

Expansion Modifiers (Relational Positioning only)

Modifier	Description
expandb2c	Hold top edge of item constant and expand bottom edge to align with ve reference point item.
expandb2t	Hold top edge of item constant and expand bottom edge to align with to point item.
expandl2c	Hold right edge of item constant and expand left edge to align with hori reference point item.
expandl2l	Hold right edge of item constant and expand left edge to align with left point item.
expandl2r	Hold right edge of item constant and expand left edge to align with righ point item.
expandr2c	Hold left edge of item constant and expand right edge to align with hori reference point item.
expandr2l	Hold left edge of item constant and expand right edge to align with left point item.
expandr2r	Hold left edge of item constant and expand right edge to align with righ point item.
expandt2b	Hold bottom edge of item constant and expand top edge to align with bo reference point item.
expandt2c	Hold bottom edge of item constant and expand top edge to align with ve reference point item.
expandt2t	Hold bottom edge of item constant and expand top edge to align with to point item.

The Extent Modifier (Relational and Absolute Positioning)

Modifier	Description
extent	Hold the top left corner of the item in place, and size the item so that it i wide as the x coordinate, and as many pixels tall as the y coordinate.

6.18.3 Available In

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

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. The absolute positioning scheme 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, to outdent the Last Name field in the above diagram, 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 either:

- Its original absolute position. For example,

```
<itemlocation content="array">
<ae content="array">
                    <ae>absolute</ae>
                    <ae>60</ae>
                    <ae>100</ae>
        </ae>
        <ae content="array">
                    <ae>offset</ae>
                    <ae>15</ae>
                    <ae>20</ae>
        </ae>
</itemlocation>
```

- Its original relational position. For example,

```
<itemlocation content="array">
        <ae content="array">
                    <ae>below</ae>
                    <ae>persInfo_label</ae>
        </ae>
        <ae content="array">
                    <ae>offset</ae>
                    <ae>15</ae>
                    <ae>20</ae>
        </ae>
</itemlocation>
```

Caution

An absolute positioning scheme is not a cross-platform solution—nor even a solution guaranteed to make forms appear the same under different video cards or in both small font and large font modes.

The sizes of many XFDL form items are measured in characters. For example, a field that is 60 x 1 in size, is 60 characters long and 1 character high. Because different platforms and video cards use differently sized fonts, even for the so-called cross-platform fonts, an item's actual size (in pixels) might change from one platform to another as the font it is measured in changes size. If relying on absolute positioning, which fastens an item to a particular pixel, to space items on a form, some items might appear overlapped on some platforms.

To ensure that forms appear the same on any platform, and under any video card or font mode, use relational positioning.

Relational Positioning Scheme

Relational 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 point items) must be defined before they can be used in an *itemlocation* statement.

When using the relational 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.

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.

6.18.4 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 content="array">

                <ae content="array">

                        <ae>absolute</ae>

                        <ae>20</ae>
        <ae>30</ae>

                </ae>

        </itemlocation>

</label>
```

Relational 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 content="array">

                <ae content="array">

                        <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 content="array">

                <ae content="array">

                        <ae>alignl2c</ae>

                        <ae>the_firm</ae>

                </ae>

                <ae content="array">

                        <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 content="array">

                <ae content="array">

                        <ae>within</ae>
                        <ae>main_toolbar</ae>
                </ae>
                <ae content="array">
                        <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 content="array">
        <ae content="array">
                <ae>absolute</ae>
                <ae>10</ae>
                <ae>10</ae>
        </ae>
        <ae content="array">
                <ae>extent</ae>
                <ae>300</ae>
                <ae>30<ae>
        </ae>
</itemlocation>
```

6.18.5 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* 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.

5. Use absolute positioning carefully. See the Caution under 6.18.3 for more information.

6.19 justify [Display]

Aligns lines of text within the space an item occupies.

6.19.1 Syntax

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

Expression	Setting	Description
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 iter

6.19.2 Available In

button, field, label, popup

6.19.3 Example

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

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

If the item’s text was:

```
The hare and the hound

went off to the woods to play
```

It would display as follows:

```
                The hare and the hound
             went off to the woods to play
```

6.19.4 Usage Notes

Default: varies depending on the item

- *button* and *popup* items: *center*
- *label* items: *left*

6.20 label[Display]

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.

6.20.1 Syntax

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

Expression	Setting	Description
label text	*string*	the text of the label

6.20.2 Available In

check, field, list, popup, radio, spacer, page global characteristics, form global characteristics

6.20.3 Example

This sample defines a typical label.

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

6.20.4 Usage Notes

1. ***Default***: none
2. The label defined in a label option has a transparent background by default. To display a particular color behind the label, set the labelbgcolor option.

3. Multiple line labels require a carriage return in the code where it is desired to appear in the label. For example:

```
<label>This label spans

two lines.</label>
```

6.21 labelbgcolor [Display]

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

6.21.1 Syntax

```
<labelbgcolor content="array">

        <ae>color name</ae>

</labelbgcolor>


<labelbgcolor content="array">

        RGB triplet

</labelbgcolor>


Note: Either format is acceptable.
```

Expression	**Setting**	**Description**
color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See 'Data Type Designators' in 9.2 f RGB triplet.

6.21.2 Available In

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

6.21.3 Examples

These samples both set the background color to red.

```
<labelbgcolor content="array">

        <ae>red</ae>

</labelbgcolor>
```

```
<labelbgcolor content="array">
        <ae>255</ae>
        <ae>0</ae>
        <ae>0</ae>
</labelbgcolor>
```

6.21.4 Usage Notes

1. Default for version 4.0.1 and greater forms: **transparent**
   This means that a label option will always be transparent unless a color is specified.

2. **Default for version 4.0.0 and lesser forms:**
   for items in the toolbar – background color of toolbar.
   for items on a page – background color of the page.

6.22 labelborderwidth[Display]

Defines the width of the border around the label specified in the *label* option. The unit of measurement is pixels.

6.22.1 Syntax

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

Expression	Setting	Description
width	*short int*	the width of the border

6.22.2 Available In

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

6.22.3 Example

This sample sets the border width to 15 pixels.

```
<labelborderwidth>15</labelborderwidth>
```

6.22.4 Usage Notes

Default: zero pixels

6.23 labelfontcolor [Display]

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

6.23.1 Syntax

```
<labelfontcolor content="array">

        <ae>color name</ae>

</labelfontcolor>


<labelfontcolor content="array">

        RGB triplet

</labelfontcolor>


Note: Either format is acceptable.
```

Expression	Setting	Description
color name	*color*	the color name
RGB triplet	*color*	the RGB triplet. See ‘Data Type Designators‘ in 9.2 f RGB triplet.

6.23.2 Available In

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

6.23.3 Examples

These samples both set the font color to green1.

```
<labelfontcolor content="array">

        <ae>green1</ae>

</labelfontcolor>


<labelfontcolor content="array">

        <ae>0</ae>

        <ae>255</ae>

        <ae>0</ae>

</labelfontcolor>
```

6.23.4 Usage Notes

Default: black

6.24 labelfontinfo[Display]

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

6.24.1 Syntax

```
<labelfontinfo content="array">

        <ae>font name</ae>

        <ae>point size</ae>

        <ae>weight</ae>

        <ae>effects</ae>

        <ae>form</ae>

</labelfontinfo>


Note: weight, effects, and form are optional.
```

Expression	**Setting**	**Description**
font name	*string*	the name of the font
point size	*short int*	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

6.24.2 Available In

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

6.24.3 Example

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

```
<labelfontinfo content="array">

        <ae>Palatino</ae>

        <ae>12</ae>
        <ae>underline</ae>
    </labelfontinfo>
```

6.24.4 Usage Notes

See the section on *fontinfo* for the usage notes.

6.25 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.

About mimedata in signature items

The *mimedata* 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 the signed hash and the signer's public key in the *mimedata* option.

6.25.1 Syntax

```
<mimedata>data</mimedata>
```

Expression	Setting	Description
data	*string*	the binary data or enclosed file contents

6.25.2 Available In

data, signature

6.25.3 Example

This sample assigns some encoded data to the *mimedata* option.

```
<mimedata>
R0lGODdhYABPAPAAAP///wAAACwAAAAAYABPAAAC/4SPqcvtD02Y
```

```
Art68+Y7im7ku2KkzXnOzh9v7qNw+k+TbDoLFTvCSPzMrS2YzmTE
</mimedata>
```

This sample shows a *mimedata* option in a digital signature.

```
<signature sid="empSignature">
        <signformat>application/x-xfdl</signformat>
        <signer>Jane D Smith, jsmith@insurance.com</signer>
        <signature>Page1.empSignature</signature>
        <signitemrefs content="array">
                <ae>omit</ae>
                <ae>Page1.mgrSigButton</ae>
                <ae>Page1.admSigButton</ae>
                <ae>Page1.empSignature</ae>
                <ae>Page1.mgrSignature</ae>
                <ae>Page1.admSignature</ae>
        </signitemrefs>
        <signoptions content="array">
                <ae>omit</ae>
                <ae>triggeritem</ae>
                <ae>coordinates</ae>
        </signoptions>
        <mimedata>MIIFMgYJKoZIhvcNACooIIFIzCCBR8CAQExDzANBgkg
        AQUFADALB\ngkqhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl
        6hMrc5DySSP+X5j\nANfBGSOI\n9w0BAQQwDwYDVQQHEwhJbn
        Rlcm5ldDEXMBUGA1UEChM\nOVmVyaVNpZ24sIEluYy4xNDAKn
        1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmRdWFsIFN1YnNjcmliy
        ZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCARExETA
        </mimedata>
</signature>
```

6.25.4 Usage Notes

1. ***Default***: none

2. Base64 encoding transforms the data into a format that can be processed easily by text editors, email applications, etc. Converting data to *base64* format ensures the resulting string contains no characters requiring an escape sequence.

3. For signatures: Because the signer's public key is included in the *mimedata*, a subsequent program can verify a signature without requiring that the signer's key be previously installed.

6.26 mimetype

Defines the MIME type of the data stored in a *data* item.

6.26.1 Syntax

```
<mimetype>MIME type</mimetype>
```

Expression	Setting	Description
MIME type	*string*	the MIME type of the data item

6.26.2 Available In

data

6.26.3 Example

This sample sets the MIME type to indicate image data.

```
<mimetype>image/gif</mimetype>
```

6.26.4 Usage Notes

1. ***Default***: application/uwi_bin

2. Here are some examples of MIME types. For full information on MIME types, read the MIME rfcs (1521, 1522 and 822). They are available on the World Wide Web.

MIME type	Meaning
application/postscript	Binary item
application/uwi_bin	Binary item
application/x-xfdl	XFDL form item
application/uwi_nothing	No data included
audio/basic	Sound item
audio/wav	Sound 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
text/richtext	Binary item
video/mpeg	Video item
video/quicktime	Video item

6.27 mouseover

Specifies whether the mouse pointer is currently over an item or page. This option is set by code outside XFDL.

6.27.1 Syntax

```
<mouseover>status</mouseover>
```

Expression	Setting	Description
status	**on**	mouse pointer is over item or page
	off	mouse pointer is not over item or page

6.27.2 Available In

button, check, combobox, field, list, popup, radio, toolbar, page settings

6.27.3 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 content="array">

                <ae content="compute">

                        <compute>mouseover=="on" ? "white" : "blue"</compute>

        </ae>

        </bgcolor>
```

```
</button>
```

6.27.4 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).

The *mouseover* option is not included in form descriptions that are saved or transmitted.

6.28 next

Identifies the item to receive focus when a user tabs ahead from the current item. If the specified item is on another page, the new page appears with the item in focus. Only modifiable items can receive focus.

6.28.1 Syntax

```
<next>item reference</next>
```

Expression	**Setting**	**Description**
item reference	*string*	identifies the item to receive focus next

6.28.2 Available In

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

6.28.3 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>
```

6.28.4 Usage Notes

1. ***Default tabbing order:***depends on the order in which page and item definitions occur within the form definition. The sequence is:

    - *first page to display*: first page defined in the form

- *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

2. 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*.
3. Placing the *next* option in form characteristics defines the first page to appear, and the first item to receive focus when the form opens. Placing *next* in page characteristics defines the first item to receive focus when the page appears.
4. If the *next* option identifies form or page characteristics, focus moves to the item defined to receive focus when the form or page appears. The form characteristics reference is *global.global*. The page characteristics reference is *global* for the current page or *page_tag.global* for another page.

6.29 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.

6.29.1 Syntax

```
<previous>item reference</previous>
```

Expression	Setting	Description
item reference	*string*	identifies the item to receive focus next

6.29.2 Available In

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

6.29.3 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>
```

6.29.4 Usage Notes

1. ***Default tabbing order:*** depends on the order in which page and item definitions occur within the form definition. The sequence is:

   - *first page to display*: first page defined in the form
   - *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 the *previous* option in form characteristics defines the first page to appear, and the first item to receive focus when the form opens. Placing *previous* in page characteristics defines the first item to receive focus when the page appears.

4. If the *previous* option identifies form or page characteristics, focus moves to the item defined to receive focus when the form or page appears. The form characteristics reference is *global.global*. The page characteristics reference is *global* for the current page or *page_tag.global* for another page.

6.30 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.

6.30.1 Syntax

```
<printsettings content="array">

        <pages>page list</pages>

        <dialog>dialog settings</dialog>

</printsettings>



Note: All settings are optional.
```

Expression	Setting	Description
page list	*(see below)*	the list of pages that to print
dialog settings	*(see below)*	determines whether the print dialog is shown, and wh be used when printing (for example, paper orientatio copies)

6.30.2 Available In

action, button, cell, page global characteristics, form global characteristics

6.30.3 Page List

The page list uses the following syntax:

```
<pages content="array">

        <ae>keep/omit</ae>

        <ae>page tag 1</ae>

        <ae>page tag 2</ae>

...

</pages>
```

The settingsfor 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.

6.30.4 Dialog Settings

The dialog settings use the following syntax

```
<dialog content="array">

        <ae>on/off</ae>

        <orientation>portrait/landscape</orientation>

        <copies>1</copies>

</dialog>
```

The settings work as follows:

Setting	Description
active	When "on", the print dialog will be displayed before the form is printed, allo change the settings. When "off", the dialog will not be shown and the form v immediately.
orientation	Determines whether the form will be printed in "landscape" or "portrait" orie
copies	Determines the number of copies that will be printed.
printpages	*See below.*

6.30.5 Example

This sample omits "page2" from printing, 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 content="array">

        <pages content="array">

                <ae>omit</ae>

                <ae>page2</ae>

        </pages>

        <dialog content="array">

                <ae>on</ae>

                <orientation>landscape</orientation>

                <copies>2</copies>

        </dialog>

</printsettings>
```

6.30.6 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.

6.31 saveformat

Specifies the format a form will be saved in. An XFDL form may be saved in XFDL format, UFDL format, or HTML format.

XFDL format saves the entire form definition, including the user input.

UFDL 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 and radio.

6.31.1 Syntax

```
<saveformat>MIME type</saveformat>
```

Expression	Setting	Description
MIME_type	**application/x-xfdl**	use XFDL forn
	application/uwi_form	use UFDL forn
	application/x-www-form-urlencoded	use HTML forı

6.31.2 Available In

button, cell, form global characteristics

6.31.3 Examples

HTML format in a button 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 global characteristics

This example shows how to use *saveformat* as a form global characteristic.

```
<?xml version="1.0"?>

<XFDL version="4.1.0">

<bgcolor content="array">

        <ae>ivory</ae>

</bgcolor>

<saveformat>application/x-xfdl</saveformat>
```

```
<page sid="page_1">
...
```

Any time a user saves this form, it will be saved in XFDL format.

6.31.4 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*.

3. ***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: *item reference* identifies the list.
popup	*item tag= value option setting of selected cell* Note: *item reference* identifies the popup.
radio	*group option setting=item tag of selected radio*
custom	*item tag=value option setting*

Note: *comboboxes cannot be saved in HTML format.*

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 (+).

```
‘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.

6.32 scrollhoriz

Defines horizontal scrolling options for a *field* item.

6.32.1 Syntax

```
<scrollhoriz>option</scrollhoriz>
```

Expression	Setting	Description
option	**never**	permit scrolling using the cursor but display no horizontal scroll l
	always	permit scrolling and display a horizontal scroll bar
	wordwrap	wrap field contents from line to line, inhibit scrolling and display horizontal scroll bar

6.32.2 Available In

field

6.32.3 Example

This sample sets the horizontal scrolling option to permit scrolling and to display the horizontal scroll bar.

```
<scrollhoriz>always</scrollhoiz>
```

6.32.4 Usage Notes

1. **Default:** *never*
2. The scroll bar displays along the field’s bottom edge.

6.33 scrollvert

Defines vertical scrolling options for a *field* item.

6.33.1 Syntax

```
<scrollvert>option</scrollvert>
```

Expression	Setting	Description
option	**never**	permit scrolling using the cursor but display no vertic
	always	permit scrolling and display a vertical scroll bar
	fixed	inhibit scrolling and display no vertical scroll bars

6.33.2 Available In

field

6.33.3 Example

This sample sets the vertical scrolling option to inhibit all scrolling.

```
<scrollvert>fixed</scrollvert>
```

6.33.4 Usage Notes

1. **Default:** *never*
2. The scroll bar displays along the field’s right edge.

6.34 signature

Used in conjunction with the button item to establish the XFDL item name by which a particular digital signature will be identified.

6.34.1 Syntax

```
<signature>name of signature</signature>
```

Expression	Setting	Description
name of signature	*string*	the name of the signature

6.34.2 Available In

button, signature

6.34.3 Example

This sample identifies the signature item for a particular button as "mysig".

```
<signature>mysig</signature>
```

6.34.4 Usage Notes

1. ***Default***: none

2. The signature option must be included in each **signature** button that is set up.

6.35 signdatagroups

Specifies which datagroups are to be filtered for a particular digital signature. (Filtering means either keeping or omitting data.) Keeping a datagroup means keeping or omitting all items containing that datagroup name, even if they were added after the form was created. This is how enclosures are signed.

6.35.1 Syntax

```
<signdatagroups content="array">

        <ae>datagroup filter</ae>

        <ae>datagroup reference1</ae>

...

        <ae>datagroup referencen</ae>

</signdatagroups>


Note: The number of datagroup reference entries is optional.
```

Expression	Setting	Description
datagroup filter	**keep**	include datagroups in the <datagroup reference> list omit those not in list
	omit	omit datagroups in the <datagroup reference> list fr include those not in list
datagroup reference	*string*	identifies a datagroup

6.35.2 Available In

button, signature

6.35.3 Example

This example specifies a *signdatagroups* option that keeps the datagroup called "Business_Letters".

```
<signdatagoups content="array">
        <ae>keep</ae>
        <ae>Business_Letters</ae>
</signdatagroups>
```

6.35.4 Usage Notes

1. ***Default***: keep
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.

6.36 signer

Identifies who signed a particular form.

6.36.1 Syntax

```
<signer>identity of user</signer>
```

Expression	Setting	Description
identity of user	*string*	identity of user

6.36.2 Available In

button, signature

6.36.3 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>
```

6.36.4 Usage Notes

1. The setting of the signer option varies, depending on where the signature is from. Using different certificate authorities might produce different results.

2. The *signer* option is automatically generated by the signature button when the user signs the form. It goes automatically into both the signature button code and the signature code. No manual coding is required.

6.37 signformat

Records the type of encoding that a Viewer 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 cryptographic service provider to use when creating a hash of the signed data.
- the type of implementation of the cryptographic service provider (for example, full implementation, only one algorithm supported, etc.)
- the algorithm to use when creating a hash of the signed data.

About the mimedata setting:

To create the *mimedata* setting, a 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 description.

6.37.1 Syntax

```
<signformat>MIMEtype;csp="csp";csptype=csptype;hashalg=alg"

</signformat>
```

Expression	Setting	Description
MIMEtype	*string*	the MIME type of the signed data. Must be either **appli** **application/uwi_form**.
csp	*string*	the cryptographic service provider to use. Must be a stri escaped double-quotation marks. The string is pre-defir API.
csptype	*string or csp-defined number*	the type of implementation of the cryptographic service allowed types, see the list in Usage Notes, below.
alg	*string*	the hash algorithm to use

6.37.2 Available In

button, signature

6.37.3 Example

```
<button sid="empSigButton">
        <type>signature</type>
        <value content="compute">
                <compute>signer</compute>
        </value>
        <signer></signer>


        <format content="array">
                <ae>string</ae>
                <ae>mandatory</ae>
        </format>
        <signformat>application/x-xfdl;csp="Microsoft Base Cryptographic
Provider v1.0";csptype=rsa_full;hashalg=sha1</signformat>
        <signoptions content="array">
                <ae>omit</ae>
                <ae>triggeritem</ae>
                <ae>coordinates</ae>
        </signoptions>


        <signitemrefs content="array">
                <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>
```

6.37.4 Usage Notes

1. A XFDL Viewer automatically copies the *signformat* option from a signature button to its associated signature item.

2. The list below describes the settings that may be used for the **csptype** parameter. (Note that a numeric value may be used, as described below the table.)

Setting	**Meaning**
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 Standar
dss_dh	For a CSP that supplies DSS compliant algorithms and Diffie-Hellman encrypti
fortezza	For a CSP that supplies Fortezza algorithms

3. Instead of using one of the settings in the table above 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.

6.38 signgroups

Specifies which groups of items are to be filtered for a particular digital signature. (Filtering means either keeping or omitting items.) Examples of grouped items are radio buttons and cells.

6.38.1 Syntax

```
<signgroups content="array">

        <ae>group filter</ae>

        <ae>group reference1</ae>

...

        <ae>group referencen</ae>

</signgroups>



Note: The number of group reference entries is optional.
```

Expression	**Setting**	**Description**
group filter	**keep**	include groups of items in the <group reference> list omit those not in list
	omit	omit groups of items in the <group reference> list frc include those not in list
group reference	*string*	identifies a group of items

6.38.2 Available In

button, signature

6.38.3 Example

This example shows a *signgroups* setting that omits the groups of items named "yesnoradiobuttons" and "monthlypaycells".

```
<signgroups content="array">
        <ae>omit</ae>
        <ae>yesnoradiobuttons</ae>
        <ae>monthlypaycells</ae>
</signgroups>
```

6.38.4 Usage Notes

1. ***Default***: keep
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.

6.39 signitemrefs

Specifies which individual items are to be filtered for a particular digital signature. (Filtering means either keeping or omitting data.)

6.39.1 Syntax

```
<signitemrefs content="array">

        <ae>item filter</ae>

        <ae>item reference1</ae>

...

        <ae>item referencen</ae>

</signitemrefs>


Note: The number of item reference entries is optional.
```

Expression	Setting	Description
item filter	**keep**	include items in the <item reference> list with the signature; o
	omit	omit items in the <item reference> list from the signature; incl list
item reference	*string*	specifies the item to be filtered

6.39.2 Available In

button, signature

6.39.3 Example

This sample sets the *signitemrefs* option to omit two fields from the digital signature.

```
<signitemrefs content="array">

        <ae>omit</ae>

        <ae>field1</ae>

        <ae>page1.field2</ae>

</signitemrefs>
```

6.39.4 Usage Notes

1. ***Default***: keep
2. Since all items have a name and type, signitemrefs filters are always applicable.
3. signitemrefs filters take precedence over signitems filters.

6.40 signitems

Specifies which types of items are to be filtered for a particular digital signature. (Filtering means either keeping or omitting data.)

6.40.1 Syntax

```
<signitems content="array">

        <ae>item filter</ae>

        <ae>item type1</ae>

...

        <ae>item typen</ae>

</signitems>


Note: The number of item type entries is optional.
```

Expression	Setting	Description
item filter	**keep**	include types of items in the <item type> list with the signature; list
	omit	omit types of items in the <item type> list from the signature; inc list
item type	*string*	specifies the type of item to be filtered

6.40.2 Available In

button, signature

6.40.3 Example

This sample sets the *signitems* option to keep the following types of items with the signature: boxes, buttons, and fields.

```
<signitems content="array">

        <ae>keep</ae>

        <ae>box</ae>

        <ae>button</ae>

        <ae>field</ae>

</signitems>
```

6.40.4 Usage Notes

1. ***Default***: keep

2. A *signitems* setting can be overridden by a *signoptions* setting, in terms of the order of precedence.

6.41 signoptionrefs

Specifies which individual options are to be filtered for a particular digital signature. (Filtering means either keeping or omitting a piece of data.) This option should be used in conjunction with a *signoptions* option also appearing in the filter.

6.41.1 Syntax

```
<signoptionrefs content="array">

        <ae>option filter</ae>

        <ae>option reference1</ae>

...

        <ae>option referencen</ae>

</signoptionrefs>


Note: The number of option reference entries is optional.
```

Expression	Setting	Description
option filter	**keep**	include options in the <option reference> list with the signat in list
	omit	omit options in the <option reference> list from the signatur not in list
option reference	*string*	specifies the option to be filtered

6.41.2 Available In

button, signature

6.41.3 Example

This example specifies a *signoptionrefs* setting that keeps a particular field with the digital signature.

```
<signoptionrefs content="array">

        <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.

6.41.4 Usage Notes

1. ***Default***: keep

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 is in valid XFDL format.

3. Signoptionrefs filters take precedence over signoptions filters.

6.42 signoptions

Specifies which types of options are to be filtered for a particular digital signature. (Filtering means either keeping or omitting a piece of data.)

6.42.1 Syntax

```
<signoptions content="array">

        <ae>option filter</ae>

        <ae>option type1</ae>

...

        <ae>option typen</ae>

</signoptions>


Note: The number of option type entries is optional.
```

Expression	Setting	Description
option filter	**keep**	include types of options in the <option type> list with the signa in list
	omit	omit types of options in the <option type> list from the signatur not in list
option type	*string*	specifies the type of option to be filtered

6.42.2 Available In

button, signature

6.42.3 Example

This example shows a *signoptions* setting that omits two types of options from the digital signature.

```
<signoptions content="array">
        <ae>omit</ae>
        <ae>url</ae>
        <ae>printsettings</ae>
</signoptions>
```

6.42.4 Usage Notes

1. ***Default***: keep
2. One signoptions setting that must always be specified is as follows:

   ```
   <signoptions content="array">
           <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.
3. *signoptions* can be overridden by a *signoptionrefs* setting.

6.43 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 *size* 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.

6.43.1 Syntax

```
<size content="array">

        <ae>width</ae>

        <ae>height</ae>

</size>
```

Expression	Setting	Description
width	*short int*	the horizontal dimension of the item
height	*short int*	the vertical dimension of the item

6.43.2 Available In

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

6.43.3 Example

This sample sets the item's size to 80 characters wide by five characters high.

```
<size content="array">

        <ae>80</ae>

        <ae>5</ae>

</size>
```

6.43.4 Usage Notes

1. ***Size and Font:*** The width might not always accommodate the number of characters specified. The calculation to determine actual width is:

   ```
   'average character width for the item's font'*width
   ```

   This 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 *line*.

3. The item and bounding box sizes can be changed by using *itemlocation* with an *expansion* or *extent* modifier.

6.44 thickness[Display]

Specifies the thickness of a *line*. The unit of measurement is pixels.

6.44.1 Syntax

```
<thickness>thickness</thickness>
```

Expression	Setting	Description
thickness	*short int*	the thickness of the line

6.44.2 Available In

line

6.44.3 Example

This sample defines a horizontal line 40 characters long and five pixels thick.

```
<size content="array">
        <ae>40</ae>
        <ae>0</ae>
        </size>
<thickness>5</thickness>
```

6.44.4 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.

6.45 transmitformat

Specifies the format of the form data submitted to a processing application. An XFDL form can submit data in XFDL format, UFDL format, or in HTML format.

XFDL format submits the entire form definition, including user input.

UFDL 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 global characteristics are sent only if the format is XFDL or UFDL.*

6.45.1 Syntax

```
<transmitformat>MIME_type</transmitformat>
```

Expression	**Setting**	**Description**
MIME_type	**application/x-xfdl**	use XFDL format
	application/uwi_form	use UFDL format
	application/x-www-form-urlencoded	use HTML form form

6.45.2 Available In

action, button, cell, form global characteristics

6.45.3 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 content="array">

                <ae>mailto:user@host.domain</ae>

        </url>

        <transmitformat>application/x-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 content="array">

                <ae>http://www.host.domain/cgi-bin/recvStatus</ae>
        </url>
        <transmitformat>application/x-www-form-urlencoded</transmitformat>
        <delay content="array">
                <ae>repeat</ae>
                <ae>180</ae>
        </delay>
</action>
```

Every 180 seconds, the form definition will be converted to HTML form format.

6.45.4 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*.

3. ***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
check	*item tag=value option setting*
field	*item tag=value option setting*
list	*item tag=value option setting of selected cell* **Note:** *item reference* identifies the list.
popup	*item tag=value option setting of selected cell* **Note:** *item reference* identifies the popup.
radio	*group option setting=item tag of selected radio*
custom	*item tag=value option setting*
all other items	*item tag=value option setting*

   Note: *comboboxes are not supported in HTML.*

Substitutions and Omissions:

- Spaces in the value are replaced by the plus sign (+).

  ```
  '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.

4. ***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.

6.46 triggeritem

Identifies the item that triggered a form submission. Items triggering form submissions have a *type* option setting of *submit* or *done*.

When a user selects an item that triggers a form submission, the *triggeritem* option is added to the form global characteristics and assigned the item reference of the selected item.

6.46.1 Syntax

```
<triggeritem>item reference</triggeritem>
```

Expression	Setting	Description
item reference	*string*	identifies the trigger item

6.46.2 Available In

form global characteristics

6.46.3 Example

This sample indicates that the item triggering the request is on the page called Page_one and has an item tag of submit_button.

```
<triggeritem>Page_one.submit_button</triggeritem>
```

7.47 type

Associates a task type with an item that can trigger a task: *action*, *button*, or *cell*.

6.47.1 Syntax

```
<type>task type</type>
```

Expression	Setting	Description
task type	*(see below)*	the task to perform

6.47.2 Task Types

The *task type* can be any of the following:

Task Type	Description of Task	Use
link	Perform all requests specified by the *url* options in the current item. See the section 'url' for more details.	acti but cell
replace	Perform a **link** followed by a **cancel**.	acti but cell
submit	Initiate the form processing applications identified in the *url* options of the current item.	acti but cell

done	Perform a **submit** followed by a **cancel**.	acti but 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.	acti but 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.	acti but cell
save	Save the form in a file specified by the user.	acti but 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	but 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.	but cell
extract	Allows a user to copy the contents of an enclosed file into a file on the local disk.	but 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.	but cell
display	Display an enclosed file. The web browser will choose the appropriate viewer according to the file's MIME type.	acti but cell
print	Print the form on a local printer.	acti but cell
signature	Create a digital signature.	but

6.47.3 Available In

action, button, cell

6.47.4 Example

This sample specifies that this item saves the form to a local file.

```
<type>save</type>
```

6.47.5 Usage Notes

Default: link

6.48 url

Identifies an object to access. 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 setting 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 setting of *submit* or *done*
  The application identified is initiated. A form processing application is an example of such an application.

- Item - used with a *type* option setting of *pagedone*
  The item identified, on the page identified, receives focus. The item must be on another page.

- Form or Page Characteristics - 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 characteristics reference is *global.global*. The page characteristics reference is <*page tag*>*.global* for another page.

6.48.1 Syntax

```
<url content="array">
        <ae>the URL1</ae>
        <ae>the URL2</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

   •  #item reference for the next item in the form to receive focus

Notes:
```

```
i) the URL can occur 1-n times.

ii) item reference can be a form or page characteristics reference.
```

Expression	Setting	Description
the URL	*string*	identifies the object to link

6.48.2 Available In

action, button, cell

6.48.3 Example

This sample identifies a form processing application.

```
<url content="array">

        <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 content="array">

        <ae>#page_2.expense_field</ae>

</url>
```

6.48.4 Usage Notes

1. ***Default***: none

2. When a form communicates with a server, the information sent might be URL-encoded. This means all non alpha-numeric characters are replaced by a character triplet consisting of the % character followed by two hexadecimal digits that form the hexadecimal value of the original character. The hexadecimal digits are "0123456789ABCDEF". For example,

Character	ASCII Number	URL-encoded
<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.

6.49 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 the status indicator; and the *value* option in *list* items contains the identity 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).

6.49.1 Syntax

```
<value>setting</value>
```

Expression	Setting	Description
setting	*string*	the item's contents

6.49.2 Available In

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

6.49.3 Example

This sample identifies the text of a *label* item.

```
<value>My Form Title</value>
```

6.49.4 Usage Notes

1. **Default:** varies by item. See the documentation for each item.

2. Multiple line values need to have carriage returns inserted in the code. For example:

```
<value>This value spans

two lines.</value>
```

3. 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.

6.50 <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.

6.50.1 Syntax

```
<custom content="array">

        <ae>expression_1</ae>

...

        <ae>expression_n</ae>



Note: <expression> can occur 1-n times.
```

Expression	Setting	Description
expression	*string*	an expression that assigns a value to the opt

6.50.2 Example

This sample shows a custom option containing an SQL query.

```
<sql_query content="array">

        <ae>SELECT NAME FROM EMPLOYEE WHERE ID = </ae>

</sql_query>
```

This statement could be included in the definition of an item that triggers a form submission. The form processing application might then complete the statement with a *value* option setting from another item, and use the statement results to populate a response form.

6.50.3 Usage Notes

The naming conventions for a custom option are as follows:

- It must begin with an alphabetic character.
- It must contain characters only from the following list: A-Z, a-z, 0-9 and underscore.
- It must contain an underscore.

7. 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.

[34] FunctionCall := (LibName '.')? FunctionName '(' (Compute (',' Compute)*)? ')'
[35] LibName ::= sid
[36] 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):

```
<status_option content="compute">

        <compute>toggle(field1.value, "high", "low")</compute>

</status_option>
```

or

```
<status_option content="compute">

        <compute>system.toggle(field1.value, "high", "low")</compute>

</status_option>
```

Calling a user-defined function (in a custom namespace)

```
<value content="compute">

        <compute>hr_funcs.holiday(field1.value, field2.value)</compute>

</value>
```

Notes on Using Functions

Position in Strings

Note that 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:

```
<str_length content="compute">

        <compute>strlen("This is a literal string")</compute>

</str_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:

```
<str_length content="compute">

        <compute>strlen(surnameField.value)</compute>

</str_length>
```

- To pass an option reference into a function (without evaluating it), surround the option reference in double-quotation marks. For example:

```
<auto_set content="compute">

        <compute>set("statusField.value", "Confirmed.")</compute>

</auto_set>
```

String Functions

7.1 countLines

7.1.1 Description

Counts the number of lines that a string **string** would take up over a given width **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.

7.1.2 Call

countLines(*string*, *width*)

7.1.3 Parameters

Expression	Setting	Description
string	*literal string or option reference*	the string to base the measurem
width	*positive int*	the width, in monospaced chara measurement on

7.1.4 Returns

The number of lines, or "" (empty) if an error occurs.

7.2 replace

7.2.1 Description

Takes a string **string** and replaces a substring in it (marked by **start** and **end**) with **newString**. 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 **newString** is not long enough (i.e., does not reach position **end**), replacement will end with the last character of **newString**. If **newString** is too long (i.e., 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.

7.2.2 Call

replace(*string, start, end, newString*)

7.2.3 Parameters

Expression	**Setting**	**Description**
string	*literal string or option reference*	the original string (enclose litera quotation marks, do not enclose quotation marks)
start	*int*	position of character at the start character in <**string**> is zero)
end	*int*	position of character at the end character in <**string**> is zero)
newString	*literal string or option reference*	the replacement string (enclose quotation marks, do not enclose quotation marks)

7.2.4 Returns

The modified string, or "" (empty) if an error occurs.

7.3 strlen

7.3.1 Description

Returns the length of **string**.

7.3.2 Call

strlen(*string*)

7.3.3 Parameters

Expression	**Setting**	**Description**
string	*literal string or option reference*	the string (enclose literal strings marks, do not enclose option ref marks)

7.3.4 Returns

A string containing the length.

7.4 strmatch

7.4.1 Description

Determines if the wildcard string wild matches the non-wildcard string real and returns the boolean result. See the *format* forms option for a description of valid wildcards.

7.4.2 Call

strmatch(*wild*, *real*)

7.4.3 Parameters

Expression	Setting	Description
wild	*literal string or option reference*	the wildcard string to match (en double quotation marks, do not in quotation marks)
real	*literal string or option reference*	the non-wildcard match string (double quotation marks, do not in quotation marks)

7.4.4 Returns

A string containing "1" if a match occurs, "0" if no match occurs.

7.5 strpbrk

7.5.1 Description

Returns the position of the first character from string2 that is found in string1.

7.5.2 Call

strpbrk(*string1, string2*)

7.5.3 Parameters

Expression	Setting	Description
string1	*literal string or option reference*	the string (enclose literal strings marks, do not enclose option ref marks)
string2	*literal string or option reference*	the string of characters (enclose quotation marks, do not enclose quotation marks)

7.5.4 Returns

A string containing the position, or "-1" if no matching characters are found.

7.6 strrstr

7.4.6.1 Description

Returns the position of the first character of the last occurrence of string2 in string1.

7.6.2 Call

strrstr(*string1, string2*)

7.6.3 Parameters

Expression	Setting	Description
string1	*literal string or option reference*	the string (enclose literal strings marks, do not enclose option ref marks)
string2	*literal string or option reference*	the substring (enclose literal stri marks, do not enclose option ref marks)

7.6.4 Returns

A string containing the position, or "-1" if no substring is found.

7.7 strstr

7.7.1 Description

Returns the position of the first character of the first occurrence of string2 in string1.

7.7.2 Call

strstr(*string1, string2*)

7.7.3 Parameters

Expression	Setting	Description
string1	*literal string or option reference*	the string (enclose literal strings marks, do not enclose option ref marks)
string2	*literal string or option reference*	the substring (enclose literal stri marks, do not enclose option ref marks)

7.7.4 Returns

A string containing the position, or "-1" if no occurrence is found.

7.8 substr

7.8.1 Description

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.

7.8.2 Call

substr(*string, start, end*)

7.8.3 Parameters

Expression	**Setting**	**Description**
string	*literal string or option reference*	the string (enclose literal strings marks, do not enclose option ref marks)
start	*int*	position of character at the start character in <**string**> is zero)
end	*int*	position of character at the end character in <**string**> is zero)

7.8.4 Returns

The substring, or "" (empty) if an error occurs.

7.9 tolower

7.9.1 Description

Returns the lower case of string.

7.9.2 Call

tolower(*string*)

7.9.3 Parameters

Expression	Setting	Description
string	*literal string or option reference*	the original string (enclose litera quotation marks, do not enclose quotation marks)

7.9.4 Returns

The lower case string.

7.10 toupper

7.10.1 Description

Returns the upper case of **string**.

7.10.2 Call

toupper(*string*)

7.10.3 Parameters

Expression	Setting	Description
string	*literal string or option reference*	the original string (enclose litera quotation marks, do not enclose quotation marks)

7.10.4 Returns

The upper case string.

7.11 trim

7.11.1 Description

Returns a copy of **string** with all leading and trailing white space (blanks, tabs, newlines, carriage returns) removed.

7.11.2 Call

trim(*string*)

7.11.3 Parameters

Expression	Setting	Description
string	*literal string or option reference*	the original string (enclose litera quotation marks, do not enclose quotation marks)

7.11.4 Returns

The string with leading and trailing whitespace removed.

7.12 URLDecode

7.12.1 Description

Returns a URL-decoded version of **string**.

7.12.2 Call

URLDecode(*string*)

7.12.3 Parameters

Expression	Setting	Description
string	*literal string or option reference*	the original string (enclose litera quotation marks, do not enclose quotation marks)

7.12.4 Returns

The URL-decoded string.

7.13 URLEncode

7.13.1 Description

Returns a URL-encoded version of **string**.

7.13.2 Call

URLEncode(*string*)

7.13.3 Parameters

Expression	Setting	Description
string	*literal string or option reference*	the original string (enclose litera quotation marks, do not enclose quotation marks)

7.13.4 Returns

The URL-encoded string.

Math Functions

7.14 abs

7.14.1 Description

Returns the absolute value of the number represented in **number**.

An error occurs if **number** is not a valid number.

7.14.2 Call

abs(*number*)

7.14.3 Parameters

Expression	Setting	Description
number	*decimal number*	a number

7.14.4 Returns

A string containing the absolute of the number, or "" if an error occurs.

7.15 acos

7.15.1 Description

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.

7.15.2 Call

acos(*number*)

7.15.3 Parameters

Expression	Setting	Description
number	*decimal number*	a number

7.15.4 Returns

A string containing the arc cosine, or "" if an error occurs.

7.16 annuity

7.16.1 Description

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.

7.16.2 Call

annuity(*rate, periods*)

7.16.3 Parameters

Expression	Setting	Description
rate	*decimal number*	the rate of interest in decimal f period
periods	*integer*	the number of periods

7.16.4 Call

A string containing the present value annuity factor, or "" if an error occurs.

7.17 asin

7.17.1 Description

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.

7.17.2 Call

asin(*number*)

7.17.3 Parameters

Expression	**Setting**	**Description**
number	*decimal number*	a number

7.17.4 Returns

A string containing the arc sine, or "" if an error occurs.

7.18 atan

7.18.1 Description

Returns the arc tangent of a number stored in **number**.

An error occurs if **number** is not a valid number.

7.18.2 Call

atan(*number*)

7.18.3 Parameters

Expression	**Setting**	**Description**
number	*decimal number*	a number

7.18.4 Returns

A string containing the arc tangent, or "" if an error occurs.

7.19 ceiling

7.19.1 Description

Returns the ceiling of the number represented in **number**.

An error occurs if **number** is not a valid number.

7.19.2 Call

ceiling(*number*)

7.19.3 Parameters

Expression	Setting	Description
number	*decimal number*	a number

7.19.4 Returns

A string containing the ceiling of the number, or "" if an error occurs.

7.20 compound

7.20.1 Description

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.

7.20.2 Call

compound(*rate*, *periods*)

7.20.3 Parameter

Expression	Setting	Description
rate	*decimal number*	the rate of interest in decimal form compound
periods	*integer*	the number of periods

7.20.4 Return

A string containing the compound interest factor, or "" if an error occurs.

7.21 cos

7.21.1 Description

Returns the cosine of an angle stored in **angle** and expressed in radians.

An error occurs if **angle** does not contain a valid angle.

7.21.2 Call

cos(*angle*)

7.21.3 Parameters

Expression	**Setting**	**Description**
angle	*decimal number*	the angle in radians

7.21.4 Call

A string containing the cosine, or "" if an error occurs.

7.22 deg2rad

7.22.1 Description

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.

7.22.2 Call

deg2rad(*angle*)

7.22.3 Parameters

Expression	**Setting**	**Description**
angle	*decimal number*	the angle in degrees

7.22.4 Returns

A string containing the number of radians, or "" if an error occurs.

7.23 exp

7.23.1 Description

Returns the exponentiation of the number represented in **number** (i.e., e$^{\textbf{number}}$).

An error occurs if **number** is not a valid number.

7.23.2 Call

exp(*number*)

7.23.3 Parameters

Expression	**Setting**	**Description**
number	*decimal number*	a number

7.23.4 Returns

A string containing the exponentiation of the number, or "" if an error occurs.

7.24 fact

7.24.1 Description

Returns the factorial value of the integer represented in **integer**.

An error occurs if **integer** is negative.

7.24.2 Call

fact(*number*)

7.24.3 Parameters

Expression	**Setting**	**Description**
integer	*integer*	a non-negative integer

7.24.4 Returns

A string containing the factorial of the integer, or "" if an error occurs.

7.25 floor

7.25.1 Description

Returns the floor of the number represented in **number**.

An error occurs if **number** is not a valid number.

7.25.2 Call

floor(*number*)

7.25.3 Parameters

Expression	Setting	Description
number	*decimal number*	a number

7.25.4 Returns

A string containing the floor of the number, or "" if an error occurs.

7.26 ln

7.26.1 Description

Returns the natural logarithm of the number represented in **number**.

An error occurs if **number** is not a decimal number greater than zero.

7.26.2 Call

ln(*number*)

7.26.3 Parameters

Expression	Setting	Description
number	*decimal number*	a number

7.26.4 Returns

A string containing the natural log of the number, or "" if an error occurs.

7.27 log

7.27.1 Description

Returns the logarithm of the number represented in **number** to the base indicated by **base**. If **base** is empty or absent, then base 10 is used.

An error occurs if either of **number** or **base** is not a valid number, or **base** is negative.

7.27.2 Call

log(*number*)

log(*number*, *base*)

7.27.3 Parameters

Expression	**Setting**	**Description**
number	*decimal number*	a number
base	*decimal number*	a number representing the base will be computed

7.27.4 Returns

A string containing the log of the number to the base, or "" if an error occurs.

7.28 mod

7.28.1 Description

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.

7.28.2 Call

mod(*number*, *divisor*)

7.28.3 Parameters

Expression	**Setting**	**Description**
number	*decimal number*	a number
divisor	*decimal number*	a number representing the divis will be computed

7.28.4 Returns

A string containing the modulus, or "" if an error occurs.

7.29 pi

7.29.1 Description

Returns the value of *PI* to the best available accuracy.

7.29.2 Call

pi()

7.29.3 Parameters

Expression	Setting	Description
(none)		

7.29.4 Returns

A string containing the value of *PI*.

7.30 power

7.30.1 Description

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.

7.30.2 Call

power(*number*, *power*)

7.30.3 Parameters

Expression	Setting	Description
number	*decimal number*	a number
power	*decimal number*	a number representing the powe will be raised

7.30.4 Returns

A string containing the number raised to the power, or "" if an error occurs.

7.31 rad2deg

7.31.1 Description

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.

7.31.2 Call

rad2deg(*angle*)

7.31.3 Parameters

Expression	Setting	Description
angle	*decimal number*	the angle in radians

7.31.4 Returns

A string containing the number of degrees, or "" if an error occurs.

7.32 rand

7.32.1 Description

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**.

7.32.2 Call

rand(*lowerlimit*, *upperlimit*)

7.32.3 Parameters

Expression	Setting	Description
lowerlimit	*integer*	the lower limit of the random number's range
upperlimit	*integer*	the upper limit of the random number's range

7.32.4 Returns

A string containing the random integer, or "" if an error occurs.

7.33 round

7.33.1 Description

Returns the number represented in **number** rounded to the nearest decimal position indicated by **place** (e.g., 100, 10, 1, 0.1, ...).

An error occurs if **number** is not a valid number or **place** is not a power of 10.

7.33.2 Call

round(*number*, *place*)

7.33.3 Parameters

Expression	**Setting**	**Description**
number	*decimal number*	a number
place	*decimal number*	a number representing the decin is to be rounded

7.33.4 Returns

A string containing the rounded number, or "" if an error occurs.

7.34 sin

7.34.1 Description

Returns the sine of an angle stored in **angle** and expressed in radians.

An error occurs if **angle** does not contain a valid angle.

7.34.2 Call

sin(*angle*)

7.34.3 Parameters

Expression	**Setting**	**Description**
angle	*decimal number*	the angle in radians

7.34.4 Returns

A string containing the sine, or "" if an error occurs.

7.35 sqrt

7.35.1 Description

Returns the square root of the number represented in **number**.

An error occurs if **number** is a negative number.

7.35.2 Call

sqrt(*number*)

7.35.3 Parameters

Expression	Setting	Description
number	*decimal number*	a non-negative number

7.35.4 Returns

A string containing the square root, or "" if an error occurs.

7.36 tan

7.36.1 Description

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, P/2, 3P/2, 5P/2, and so on).

7.36.2 Call

tan(*angle*)

7.36.3 Parameters

Expression	Setting	Description
angle	*decimal number*	the angle in radians

7.36.4 Returns

A string containing the tangent, or "" if an error occurs.

Utility Functions

7.37 applicationName

7.37.1 Description

Returns the name of the currently running application.

7.37.2 Call

applicationName()

7.37.3 Parameters

Expression	Setting	Description
(none)		

7.37.4 Returns

A string containing the application name.

7.38 applicationVersion

7.38.1 Description

Returns the version of the currently running application in the format "MM.mm.TT".

7.38.2 Call

applicationVersion()

7.38.3 Parameters

Expression	Setting	Description
(none)		

7.38.4 Returns

A string containing the application version.

7.39 applicationVersionNum

7.39.1 Description

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.

7.39.2 Call

applicationVersionNum()

7.39.3 Parameters

Expression	Setting	Description
(none)		

7.39.4 Returns

A string containing the application version number.

7.40 decimal

7.40.1 Description

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**.

7.40.2 Call

decimal(*number*, *base*)

7.40.3 Parameters

Expression	Setting	Description
number	*number*	a number
base	*positive integer*	an integer that is the base of the

7.40.4 Returns

A string containing the decimal representation of the number, or "" if an error occurs.

7.41 formatString

7.41.1 Description

Returns a string string formatted according to the rules set out in the referenced *format* option formatOptionReference.

An error occurs if an invalid format is specified.

7.41.2 Call

formatString(*string*, *itemtagOfFormat*)

7.41.3 Parameters

Expression	Setting	Description
string	*a string*	a string to format acc option
formatOptionReference	*an option reference, including the page tag, if necessary*	the option reference c when in formatting th

7.41.4 Returns

The formatted string.

7.42 isValidFormat

7.42.1 Description

Returns the boolean result of whether a string **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.

7.42.2 Call

isValidFormat(*string*, *formatOptionReference*)

7.42.3 Parameters

Expression	Setting	Description
string	*a string*	a string to be checked agair
formatOptionReference	*an option reference, including the page tag, if necessary*	the option reference of the against

7.42.4 Call

"1" if the string follows the format, "0" if not, or "" if an error occurs.

7.43 set

7.43.1 Description

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.

An error occurs if the specified form option could not be set to the specified value.

7.43.2 Call

set(*reference, value*)

7.43.3 Parameters

Expression	Setting	Description
reference	*form option reference*	an adequately qualified referenc
value	*form value*	a string containing the option's

7.43.4 Returns

"1" if the operation completed successfully, "0" if an error occurred.

7.44 toggle

7.44.1 Description

Detects transitions in a form option specified by **reference**, and returns a result. If *toggle* contains just a **reference** parameter, then *toggle* returns "**1**" every time the referenced setting changes. If *toggle* contains the **reference** parameter and the **from** and **to** parameters, then *toggle* returns "**1**" when the setting changes from the **from** state to the **to** state, and "**0**" at other times.

An error occurs if the specified form option does not exist.

7.44.2 Call

toggle(*reference*)

toggle(*reference, from, to*)

7.44.3 Parameters

Expression	Setting	Description
reference	*form option reference*	an adequately qualified referenc
from	*form value*	a string containing a possible op
to	*form value*	a string containing a possible op

7.44.4 Returns

"1" if the specified change occurs in the specified option, or "0" if another change occurs.

Time and Date Functions

7.45 date

7.45.1 Description

Returns the current date in "yyyymmdd" format.

7.45.2 Call

date()

7.45.3 Parameters

Expression	Setting	Description
(none)		

7.45.4 Returns

A string containing the current date.

7.46 dateToSeconds

7.46.1 Description

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.

An error occurs if either of date or time is not well-formed.

7.46.2 Call

dateToSeconds(*date*, *time*)

7.46.3 Parameters

Expression	Setting	Description
date	*date string*	a date in a recognized format
time	*time string*	a time in a recognized format

7.46.4 Returns

A string containing the number of seconds, or "" if an error occurs.

7.47 day

7.47.1 Description

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.

7.47.2 Call

day(*dateSecs*|"")

7.47.3 Parameters

Expression	**Setting**	**Description**
dateSecs	*number*	a date represented by the number of seconds s January 1, 1970

7.47.4 Returns

A string containing the day, or "" if an error occurs.

7.48 dayOfWeek

7.48.1 Description

Returns the numeric day of the week (Sunday=1, etc.) 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.

7.48.2 Call

dayOfWeek(*dateSecs*|"")

7.48.3 Parameters

Expression	Setting	Description
dateSecs	*number*	a date represented by the number of seconds sin January 1st, 1970

7.48.4 Returns

A string containing the day of the week, or "" if an error occurs.

7.49 endOfMonth

7.49.1 Description

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.

7.49.2 Call

endOfMonth(*dateSecs*|"")

7.49.3 Parameters

Expression	Setting	Description
dateSecs	*number*	a date represented by the number of seconds si January 1st, 1970

7.49.4 Returns

A string containing the number of seconds, or "" if an error occurs.

7.50 hour

7.50.1 Description

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.

7.50.2 Call

hour(*dateSecs*|"")

7.50.3 Parameters

Expression	Setting	Description
dateSecs	*number*	a date represented by the number of seconds January 1st, 1970

7.50.4 Returns

A string containing the hour, or "" if an error occurs.

7.51 minute

7.51.1 Description

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.

7.51.2 Call

minute(*dateSecs*|"")

7.51.3 Parameters

Expression	Setting	Description
dateSecs	*number*	a date represented by the number of seconds sin January 1st, 1970

7.51.4 Returns

A string containing the minute, or "" if an error occurs.

7.52 month

7.52.1 Description

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.

7.52.2 Call

month(*dateSecs*|"")

7.52.3 Parameters

Expression	**Setting**	**Description**
dateSecs	*number*	a date represented by the number of seconds si January 1st, 1970

7.52.4 Returns

A string containing the month, or "" if an error occurs.

7.53 now

7.53.1 Description

Returns the number of seconds since 00:00:00 GMT, January 1st, 1970.

7.53.2 Call

now()

7.53.3 Parameters

Expression	**Setting**	**Description**
(none)		

7.53.4 Returns

A string containing the number of seconds.

7.54 second

7.54.1 Description

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.

7.54.2 Call

second(*dateSecs*|"")

7.54.3 Parameters

Expression	**Setting**	**Description**
dateSecs	*number*	a date represented by the number of seconds since 1st, 1970

7.54.4 Returns

A string containing the second, or "" if an error occurs.

7.55 time

7.55.1 Description

Returns the current time in "hh:mm:AM" format.

7.55.2 Call

time()

7.55.3 Parameters

Expression	**Setting**	**Description**
(none)		

7.55.4 Returns

A string containing the current time.

7.56 year

7.56.1 Description

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.

7.56.2 Call

year(*dateSecs*|"")

7.56.3 Parameters

Expression	**Setting**	**Description**
dateSecs	*number*	a date represented by the number of seconds sin January 1st, 1970

7.56.4 Returns

A string containing the year, or "" if an error occurs.

Appendix A: Color Table

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 content="array">

        <ae>255</ae>

        <ae>255</ae>

        <ae>255</ae>

</bgcolor>
```

is equivalent to:

```
<bgcolor content="array">

        <ae>white</ae>

</bgcolor>
```

Note: The **transparent** color has no RGB equivalent.

The following table lists the names and RGB triplet values for the available colors:

R G B	COLOR NAME	R G B	COLOR NAME
240 248 255	AliceBlue	139 134 78	khaki4
250 235 215	AntiqueWhite	230 230 250	lavender
255 239 219	AntiqueWhite1	255 240 245	LavenderBlush
238 223 204	AntiqueWhite2	255 240 245	LavenderBlush1
205 192 176	AntiqueWhite3	238 224 229	LavenderBlush2
139 131 120	AntiqueWhite4	205 193 197	LavenderBlush3
127 255 212	aquamarine1	139 131 134	LavenderBlush4
118 238 198	aquamarine2	124 252 0	LawnGreen
102 205 170	aquamarine3	255 250 205	LemonChiffon
69 139 116	aquamarine4	255 250 205	LemonChiffon1
240 255 255	azure1	238 233 191	LemonChiffon2
224 238 238	azure2	205 201 165	LemonChiffon3
193 205 205	azure3	139 137 112	LemonChiffon4
131 139 139	azure4	173 216 230	LightBlue
245 245 220	beige	191 239 255	LightBlue1

255 228 196	bisque1	178 223 238	LightBlue2
238 213 183	bisque2	154 192 205	LightBlue3
205 183 158	bisque3	104 131 139	LightBlue4
139 125 107	bisque4	240 128 128	LightCoral
0 0 0	black	224 255 255	LightCyan
255 235 205	BlanchedAlmond	224 255 255	LightCyan1
0 0 255	blue	209 238 238	LightCyan2
0 0 255	blue1	180 205 205	LightCyan3
0 0 238	blue2	122 139 139	LightCyan4
0 0 205	blue3	238 221 130	LightGoldenrod
0 0 139	blue4	255 236 139	LightGoldenrod1
138 43 226	BlueViolet	238 220 130	LightGoldenrod2
165 42 42	brown	205 190 112	LightGoldenrod3
255 64 64	brown1	139 129 76	LightGoldenrod4
238 59 59	brown2	250 250 210	LightGoldenrodYellow
205 51 51	brown3	211 211 211	LightGray

139 35 35	brown4	255 182 193	LightPink
222 184 135	burlywood	255 174 185	LightPink1
255 211 155	burlywood1	238 162 173	LightPink2
238 197 145	burlywood2	205 140 149	LightPink3
205 170 15	burlywood3	139 95 101	LightPink4
139 115 85	burlywood4	255 160 122	LightSalmon
95 158 160	CadetBlue	255 160 122	LightSalmon1
152 245 255	CadetBlue1	238 149 114	LightSalmon2
142 229 238	CadetBlue2	205 129 98	LightSalmon3
122 197 205	CadetBlue3	139 87 66	LightSalmon4
83 134 19	CadetBlue4	32 178 170	LightSeaGreen
127 255 0	chartreuse1	135 206 250	LightSkyBlue
118 238 0	chartreuse2	176 226 255	LightSkyBlue1
102 205 0	chartreuse3	164 211 238	LightSkyBlue2
69 139 0	chartreuse4	141 182 205	LightSkyBlue3
210 105 30	chocolate	96 123 139	LightSkyBlue4

255 127 36	chocolate1	132 112 255	LightSlateBlue
238 118 33	chocolate2	119 136 153	LightSlateGray
205 102 29	chocolate3	176 196 222	LightSteelBlue
139 69 19	chocolate4	202 225 255	LightSteelBlue1
255 127 80	coral	188 210 238	LightSteelBlue2
255 114 86	coral1	162 181 205	LightSteelBlue3
238 106 80	coral2	110 123 139	LightSteelBlue4
205 91 69	coral3	255 255 224	LightYellow
139 62 47	coral4	255 255 224	LightYellow1
100 149 237	CornflowerBlue	238 238 209	LightYellow2
255 248 220	cornsilk1	205 205 180	LightYellow3
238 232 205	cornsilk2	139 139 122	LightYellow4
205 200 177	cornsilk3	50 205 50	LimeGreen
139 136 120	cornsilk4	250 240 230	linen
0 255 255	cyan1	255 0 255	magenta1
0 238 238	cyan2	238 0 238	magenta2

0 205 205	cyan3	205 0 205	magenta3
0 139 139	cyan4	139 0 139	magenta4
184 134 11	DarkGoldenrod	176 48 96	maroon
255 185 15	DarkGoldenrod1	255 52 179	maroon1
238 173 14	DarkGoldenrod2	238 48 167	maroon2
205 149 12	DarkGoldenrod3	205 41 144	maroon3
139 101 8	DarkGoldenrod4	139 28 98	maroon4
0 100 0	DarkGreen	102 205 170	MediumAquamarine
189 183 107	DarkKhaki	0 0 205	MediumBlue
85 107 47	DarkOliveGreen	186 85 211	MediumOrchid
202 255 112	DarkOliveGreen1	224 102 255	MediumOrchid1
188 238 104	DarkOliveGreen2	209 95 238	MediumOrchid2
162 205 90	DarkOliveGreen3	180 82 205	MediumOrchid3
110 139 61	DarkOliveGreen4	122 55 139	MediumOrchid4
255 140 0	DarkOrange	147 112 219	MediumPurple
255 127 0	DarkOrange1	171 130 255	MediumPurple1

238 118 0	DarkOrange2	159 121 238	MediumPurple2
205 102 0	DarkOrange3	137 104 205	MediumPurple3
139 69 0	DarkOrange4	93 71 139	MediumPurple4
153 50 204	DarkOrchid	60 179 113	MediumSeaGreen
191 62 255	DarkOrchid1	123 104 238	MediumSlateBlue
178 58 238	DarkOrchid2	0 250 154	MediumSpring Green
154 50 205	DarkOrchid3	72 209 204	MediumTurquoise
104 34 139	DarkOrchid4	199 21 133	MediumVioletRed
233 150 122	DarkSalmon	25 25 112	MidnightBlue
143 188 143	DarkSeaGreen	245 255 250	MintCream
193 255 193	DarkSeaGreen1	255 228 225	MistyRose
180 238 180	DarkSeaGreen2	255 228 225	MistyRose1
155 205 155	DarkSeaGreen3	238 213 210	MistyRose2
105 139 105	DarkSeaGreen4	205 183 181	MistyRose3
72 61 139	DarkSlateBlue	139 125 123	MistyRose4
47 79 79	DarkSlateGray	255 228 181	moccasin

151 255 255	DarkSlateGray1	255 222 173	NavajoWhite
141 238 238	DarkSlateGray2	255 222 173	NavajoWhite1
121 205 205	DarkSlateGray3	238 207 161	NavajoWhite2
82 139 139	DarkSlateGray4	205 179 139	NavajoWhite3
0 206 209	DarkTurquoise	139 121 94	NavajoWhite4
148 0 211	DarkViolet	0 0 128	NavyBlue
255 20 147	DeepPink1	253 245 230	OldLace
238 18 137	DeepPink2	107 142 35	OliveDrab
205 16 118	DeepPink3	192 255 62	OliveDrab1
139 10 80	DeepPink4	179 238 58	OliveDrab2
0 191 255	DeepSkyBlue1	154 205 50	OliveDrab3
0 178 238	DeepSkyBlue2	105 139 34	OliveDrab4
0 154 205	DeepSkyBlue3	255 165 0	orange1
0 104 139	DeepSkyBlue4	238 154 0	orange2
105 105 105	DimGrey	205 133 0	orange3
30 144 255	DodgerBlue1	139 90 0	orange4

28 134 238	DodgerBlue2	255 69 0	OrangeRed1
24 116 205	DodgerBlue3	238 64 0	OrangeRed2
16 78 139	DodgerBlue4	205 55 0	OrangeRed3
178 34 34	firebrick	139 37 0	OrangeRed4
255 48 48	firebrick1	218 112 214	orchid
238 44 44	firebrick2	255 131 250	orchid1
205 38 38	firebrick3	238 122 233	orchid2
139 26 26	firebrick4	205 105 201	orchid3
255 250 240	FloralWhite	139 71 137	orchid4
34 139 34	ForestGreen	238 232 170	PaleGoldenrod
220 220 220	gainsboro	152 251 152	PaleGreen
248 248 255	GhostWhite	154 255 154	PaleGreen1
255 215 0	gold1	144 238 144	PaleGreen2
238 201 0	gold2	124 205 124	PaleGreen3
205 173 0	gold3	84 139 84	PaleGreen4
139 117 0	gold4	175 238 238	PaleTurquoise

218 165 32	goldenrod	187 255 255	PaleTurquoise1
255 193 37	goldenrod1	174 238 238	PaleTurquoise2
238 180 34	goldenrod2	150 205 205	PaleTurquoise3
205 155 29	goldenrod3	102 139 139	PaleTurquoise4
139 105 20	goldenrod4	219 112 147	PaleVioletRed
192 192 192	gray	255 130 171	PaleVioletRed1
0 0 0	gray0	238 121 159	PaleVioletRed2
3 3 3	gray1	205 104 137	PaleVioletRed3
26 26 26	gray10	139 71 93	PaleVioletRed4
255 255 255	gray100	255 239 213	PapayaWhip
28 28 28	gray11	255 218 185	PeachPuff
31 31 31	gray12	255 218 185	PeachPuff1
33 33 33	gray13	238 203 173	PeachPuff2
36 36 36	gray14	205 175 149	PeachPuff3
38 38 38	gray15	139 119 101	PeachPuff4
41 41 41	gray16	205 133 63	peru

43 43 43	gray17	255 192 203	pink
46 46 46	gray18	255 181 197	pink1
48 48 48	gray19	238 169 184	pink2
5 5 5	gray2	205 145 158	pink3
51 51 51	gray20	139 99 108	pink4
54 54 54	gray21	221 160 221	plum
56 56 56	gray22	255 187 255	plum1
59 59 59	gray23	238 174 238	plum2
61 61 61	gray24	205 150 205	plum3
64 64 64	gray25	139 102 139	plum4
66 66 66	gray26	176 224 230	PowderBlue
69 69 69	gray27	160 32 240	purple
71 71 71	gray28	155 48 255	purple1
74 74 74	gray29	145 44 238	purple2
8 8 8	gray3	125 38 205	purple3
77 77 77	gray30	85 26 139	purple4

79 79 79	gray31	255 0 0	red1
82 82 82	gray32	238 0 0	red2
84 84 84	gray33	205 0 0	red3
87 87 87	gray34	139 0 0	red4
89 89 89	gray35	188 143 143	RosyBrown
92 92 92	gray36	255 193 193	RosyBrown1
94 94 94	gray37	238 180 180	RosyBrown2
97 97 97	gray38	205 155 155	RosyBrown3
99 99 99	gray39	139 105 105	RosyBrown4
10 10 10	gray4	65 105 225	RoyalBlue
102 102 102	gray40	72 118 255	RoyalBlue1
105 105 105	gray41	67 110 238	RoyalBlue2
107 107 107	gray42	58 95 205	RoyalBlue3
110 110 110	gray43	39 64 139	RoyalBlue4
112 112 112	gray44	139 69 19	SaddleBrown
115 115 115	gray45	250 128 114	salmon

117 117 117	gray46	255 140 105	salmon1
120 120 120	gray47	238 130 98	salmon2
122 122 122	gray48	205 112 84	salmon3
125 125 125	gray49	139 76 57	salmon4
13 13 13	gray5	244 164 96	SandyBrown
127 127 127	gray50	46 139 87	SeaGreen
130 130 130	gray51	84 255 159	SeaGreen1
133 133 133	gray52	78 238 148	SeaGreen2
135 135 135	gray53	67 205 128	SeaGreen3
138 138 138	gray54	46 139 87	SeaGreen4
140 140 140	gray55	255 245 238	seashell1
143 143 143	gray56	238 229 222	seashell2
145 145 145	gray57	205 197 191	seashell3
148 148 148	gray58	139 134 130	seashell4
150 150 150	gray59	160 82 45	sienna
15 15 15	gray6	255 130 71	sienna1

153 153 153	gray60	238 121 66	sienna2
156 156 156	gray61	205 104 57	sienna3
158 158 158	gray62	139 71 38	sienna4
161 161 161	gray63	135 206 235	SkyBlue
163 163 163	gray64	135 206 255	SkyBlue1
166 166 166	gray65	126 192 238	SkyBlue2
168 168 168	gray66	108 166 205	SkyBlue3
171 171 171	gray67	74 112 139	SkyBlue4
173 173 173	gray68	106 90 205	SlateBlue
176 176 176	gray69	131 111 255	SlateBlue1
18 18 18	gray7	122 103 238	SlateBlue2
179 179 179	gray70	105 89 205	SlateBlue3
181 181 181	gray71	71 60 139	SlateBlue4
184 184 184	gray72	112 128 144	SlateGray
186 186 186	gray73	198 226 255	SlateGray1
189 189 189	gray74	185 211 238	SlateGray2

191 191 191	gray75	159 182 205	SlateGray3
194 194 194	gray76	108 123 139	SlateGray4
196 196 196	gray77	255 250 250	snow1
199 199 199	gray78	238 233 233	snow2
201 201 201	gray79	205 201 201	snow3
20 20 20	gray8	139 137 137	snow4
204 204 204	gray80	0 255 127	SpringGreen1
207 207 207	gray81	0 238 118	SpringGreen2
209 209 209	gray82	0 205 102	SpringGreen3
212 212 212	gray83	0 139 69	SpringGreen4
214 214 214	gray84	70 130 180	SteelBlue
217 217 217	gray85	99 184 255	SteelBlue1
219 219 219	gray86	92 172 238	SteelBlue2
222 222 222	gray87	79 148 205	SteelBlue3
224 224 224	gray88	54 100 139	SteelBlue4
227 227 227	gray89	210 180 140	tan

23 23 23	gray9	255 165 79	tan1
229 229 229	gray90	238 154 73	tan2
232 232 232	gray91	205 133 63	tan3
235 235 235	gray92	139 90 43	tan4
237 237 237	gray93	216 191 216	thistle
240 240 240	gray94	255 225 255	thistle1
242 242 242	gray95	238 210 238	thistle2
245 245 245	gray96	205 181 205	thistle3
247 247 247	gray97	139 123 139	thistle4
250 250 250	gray98	255 99 71	tomato1
252 252 252	gray99	238 92 66	tomato2
0 255 0	green1	205 79 57	tomato3
0 238 0	green2	139 54 38	tomato4
0 205 0	green3	64 224 208	turquoise
0 139 0	green4	0 245 255	turquoise1
173 255 47	GreenYellow	0 229 238	turquoise2

240 255 240	honeydew1	0 197 205	turquoise3
224 238 224	honeydew2	0 134 139	turquoise4
193 205 193	honeydew3	238 130 238	violet
131 139 131	honeydew4	208 32 144	VioletRed
255 105 180	HotPink	255 62 150	VioletRed1
255 110 180	HotPink1	238 58 140	VioletRed2
238 106 167	HotPink2	205 50 120	VioletRed3
205 96 144	HotPink3	139 34 82	VioletRed4
139 58 98	HotPink4	245 222 179	wheat
205 92 92	IndianRed	255 231 186	wheat1
255 106 106	IndianRed1	238 216 174	wheat2
238 99 99	IndianRed2	205 186 150	wheat3
205 85 85	IndianRed3	139 126 102	wheat4
139 58 58	IndianRed4	255 255 255	white
255 255 240	ivory1	245 245 245	WhiteSmoke
238 238 224	ivory2	255 255 0	yellow

205 205 193	ivory3	255 255 0	yellow1
139 139 131	ivory4	238 238 0	yellow2
255 246 143	khaki1	205 205 0	yellow3
238 230 133	khaki2	139 139 0	yellow4
205 198 115	khaki3	154 205 50	YellowGreen