INTERNET DRAFT
David Manning, Richard Bennett, John Boyer,
Sonja McLellan, Michael Mansell
August 1998
Expires: February 04, 1999

Universal Forms Description Language Specification
Version 4.0.1
<draft-gordon-ufdl-spec-02.txt>

Status of this Memo

Abstract

The Universal Forms Description Language (UFDL) describes complex business forms for use over the Internet. The objective of the UFDL is to enable the creation of cross-platform Internet business forms that (1) contain both the complex logic and precise layout that administrators require, (2) are simple to maintain and distribute, and (3) integrate easily with existing business systems. As more and more business is done over the Internet, the need for a form description language that incorporates these features grows. HTML is designed for Internet pages, and is severely limited as a form language. This document specifies the vocabulary, syntax, and meaning of the UFDL.

CONTENTS

1. INTRODUCTION

1.1 Introduction to the UFDL

This document specifies the Universal Forms Description Language (UFDL), which describes complex business forms for use over the Internet. The objective of the UFDL is to enable the creation of

cross-platform Internet business forms that (1) contain both the complex logic and precise layout that administrators require, (2) are simple to maintain and distribute, and (3) integrate easily with existing business systems. This document specifies the vocabulary,

syntax, and meaning of the UFDL.

Since more and more business is being done over the Internet, the need for a form description language that incorporates the complexities of business systems is growing. Typically, an electronic business form is part of a process-intensive administration system. Users or server modules populate forms with data, the forms are distributed according to a work flow plan, and the data is stored in a database (or, in departments that have no complete electronic solution, the form is printed for storage). The forms, which can contain hundreds of input items, need to validate the data they receive, perform calculations and other logical operations, and integrate with existing data management systems. Today, most Internet forms are inadequate and are being created with HTML.

HTML is designed for the easy display of Internet pages. As a result, HTML is very good at creating the layout for web sites and has become the standard for web pages. Web designers and IS organizations are now trying to push HTML beyond what it was intended to do. HTML forms work well for collecting basic information over the Internet. However, most business forms are much more complex than the typical HTML order form.

HTML was not designed to collect, validate, manipulate, or store information. In order to build significant intelligence into an HTML form, a developer has to use JavaScript. Business forms also may need to travel through nodes in distribution chains, being viewed or changed by people along the way. HTML forms submit merely the data they've collected-the user interface and intelligence don't accompany it, and so make it difficult to create a workflow system for the form. HTML forms also have a fairly inflexible layout, and it's impossible to create precise, complex HTML forms and print them the way people are used to.

The UFDL was designed specifically for Internet business forms. It describes all components of a complex form: user interface, intelligent features, and input data. A UFDL form can be transmitted whole or in part from node to node in a distribution chain. The UFDL's precise layout specifications allow users to create and print forms that replicate the paper forms they're used to. The UFDL includes complex business logic so that intelligent features like user-input checking, calculations, and in-form decisions are part of the form itself, rather than a separate script, and travel with the form to the next user. The UFDL allows developers to extend the language to interface with other applications by adding their own customized information to forms. The syntax of the UFDL is high-level and easy to learn, but at the same time incorporates the logic needed for business transactions. C and Java programmers will recognize many features of the syntax.

1.2 UFDL Documentation

This section outlines how this document is organized, and directs

readers to other documents on the Universal Forms Description Language for further information.

1.2a How This Document is Organized

The UFDL Specification is intended both for an academic audience and for form developers and people writing applications that use UFDL forms.

For an introduction to the language and its elements, see Part 2: Introduction to the Universal Forms Description Language. It explains the concepts behind the UFDL and specifies the components of a UFDL form. It delineates the UFDL syntax and explains the language elements.

For a full description of form global settings, form items, form options, and directives for form viewers, see parts 2, 3, 4, and 5.

For the Backus-Naur Form (BNF) of the UFDL, see 'Appendix A: Grammar of the UFDL'. C Programmers may find it useful to review 'Appendix C: UFDL for C and C++ Programmers'.

1.2b Other UFDL Documentation

Those who want to find out more about the grammar behind the UFDL may want to view or download the Lexical and Syntactical Specification for the UFDL.

Both of these documents are available at http://www.uwi.com/UFDL

1.3 Requirement Levels for UFDL Elements

This specification does not contain extraneous material, and therefore most implementers of the UFDL will want to include all elements specified here. However, not all elements are required, though all are suggested.
This section specifies which elements are REQUIRED, RECOMMENDED, and OPTIONAL in an implementation. The criterion for determining whether an element of the language is REQUIRED is whether the exclusion of the element would prevent people from filling and transmitting the form.
Unless specified in the list below, all elements are REQUIRED. An implementation that does not include an element MUST interoperate with another implementation that does include the element (though perhaps with reduced functionality). In the same vein, an implementation that does include the element MUST interoperate with one that does not (except, of course, for the feature the element provides). Also, before deciding to ignore an element that is RECOMMENDED, an implementor must understand the implications of not including the element.

RECOMMENDED Elements (Elements that implementors SHOULD include)
- bgcolor option
- fontcolor option

- labelbgcolor option
- labelfontcolor option
- next option
- previous option
- printsettings option

OPTIONAL Elements (Elements that implementors MAY include)

- help item

- bordercolor option
- borderwidth option
- help option
- labelbordercolor option
- labelborderwidth option

- #include directive

Note: For a definition of the words REQUIRED, RECOMMENDED, OPTIONAL, MUST, SHOULD, and MAY as used in this section, see RFC 2119.

1.4 Implied Semantics for UFDL Viewers

There are a few behaviors that are "implied" but not explicit in the UFDL, and that are defining features of the UFDL. This section outlines those behaviors, and should be considered part of the UFDL Specification.

Temporary Files

A viewer that uses UFDL forms may create temporary files in the following locations:

- web browser's temp directory
- Windows temp directory
- viewer's temp directory

A viewer MUST NOT create temporary files in any other location on a user's computer. This prevents system files or permanent user files from being at risk if they're not in temp directories. A viewer may delete files from the three temporary directories listed above at its discretion, but it MUST delete ONLY files that are older than the last reboot of the operating system, or that it can positively identify as one of its own temporary files. The following UFDL form events may cause a UFDL viewer to create and/or delete temporary files: Opening a form; Closing a form; Submitting a form (a transaction of type "submit" or "done"); Emailing a form (if a viewer supports emailing forms); Enclosing files; Displaying enclosures.

Permanent Files

Certain UFDL form operations require a viewer to read or create permanent files. They are: Enclosing a File; Extracting a File; and Saving a form. Only button and cell items can initiate these operations. Automatic actions MUST NOT initiate actions that create permanent files on a user's computer.

When a viewer performs an enclose, extract, or save operation, it MUST conform to the restrictions that follow.

Enclosures: When the user activates an enclose button or cell, the viewer must prompt the user with a file browser so that the user can choose which file to enclose. This file browser must allow the

user to cancel the enclose transaction without writing the enclosure into the form. Users may choose to enclose any files to which their operating system gives them access.

Extractions: When the user activates an extract button or cell, the viewer must prompt the user with a file browser so that the user may choose both a location and a name for the file that's being extracted. Other than the usual restrictions on file names that the user's operating system imposes, the viewer must not restrict the file name the user chooses. If the user specifies a file name that already exists, then the viewer must warn the user that it exists, and ask the user whether to overwrite the existing file. The user must be able to cancel the extract operation before the viewer has written the permanent file.

Saves: When the user activates a save button or cell, the viewer must prompt the user with a file browser so that the user may choose both a location and a name for the saved form. (Save acts like "Save As".) Other than the usual restrictions on file names that the user's operating system imposes, the viewer must not restrict the file name the user chooses. If there is already a file with the file name that the user specifies, then the viewer must warn the user that it exists, and ask the user whether to overwrite the existing file. The viewer must allow the user to cancel the save operation before the viewer has written the permanent file.

These rules have been created in order to allow users to perform the enclosures, extractions, and saves necessary when completing business forms, while at the same time protecting their computers by (a) limiting temporary files to temp directories, and (b) preventing uploads and downloads that users are not aware of.

1.5 Security Considerations

The UFDL specifies the description of a form, but not the transport protocol for transmitting it. Any trasmission security issues that exist for the transport protocol submitting the form (for example, those used by mail programs and web browsers) exist when transmitting a UFDL form. (Note, however, that UFDL forms can be compressed using a compression algorithm before they are submitted. For more information, see the transmitformat option description.)

UFDL forms cannot invoke programs on local computer drives. In addition, a UFDL viewer must save temporary files to standard temp directories only, as outlined in '1.4 Implied Semantics' above. A UFDL Viewer may only read and write permament files under strict conditions and then only with the user's knowledge (through presenting a file browser); see '1.4 Implied Semantics' for more information.

1.6 Responding to Errors in the Form Description

Any UFDL form interpreter must parse a UFDL form for non-compliance to the UFDL specification. This debugger should treat

non-compliances in the following manner:

Flag as Warnings - All item types and option types that are not part of the UFDL. These must be flagged as warnings and not as errors because the UFDL allows developers to create custom items and options for inserting application-specific information into forms. Forms containing non-compliances that generate warning messages may still be displayed. The non-compliances must be ignored when displaying the form, and the defaults used instead (if applicable). A UFDL Viewer may implement a mechanism that allows users to turn off the warning messages.

Flag as Errors - Anything that might (but also might not) adversely affect the appearance or functionality of the form. Forms that contain non-compliances that might affect the appearance or functionality of the form may be displayed. The non-compliances must be ignored, and the defaults (if applicable) must be used when displaying the form.

Flag as Fatal Errors - Anything that will adversely affect the appearance or functionality of the form. Forms containing non-compliances that generate fatal error messages must not be displayed.

In addition, the UFDL debugger must check the version number of the form it parses. The version number denotes which version of the UFDL specification the form complies with. The parser must check for non-compliances based on the version of the UFDL that the form was written with. This provides backwards compatibility.

---

2. Introduction to the Universal Forms Description Language

2.1 What is UFDL?

Summary

The Universal Forms Description Language (UFDL) is a language that describes complex Internet business forms much the way HTML describes web pages. It is cross-platform, easy to learn, and its features are tailored to business needs.
Note: Because UFDL version 4.0 includes the start value element in an option name, any code written to work with the UFDL BNF version 3.3.1 or earlier will not be able to parse a version 4.0 form.

Details

UFDL is a platform-independent, high-level language that describes Internet business forms. It was designed specifically for creating forms that are capable of replacing paper forms

systems. That is, it creates forms that:

- Create auditable records, by viewing a form as an object that

includes layout instructions and data, and that can be passed whole from node to node in a distribution chain, archived, and retrieved later for verification.
- Let users work offline or online.
- Perform logical operations, functions, and other behavioral changes based on user events.
- Give users editing and error checking tools.
- Allow users to digitally sign the whole form or parts of the form.
- Appear the same on any platform and under any screen resolution and system font size.
- Interface with other applications.

UFDL incorporates the following design concepts:

Familiar Syntax

UFDL is easy to pick up, because it is syntactically similar to two industry standard programming languages: C++ and Java. Here is the description of a very simple UFDL form:

```
version = "3.2.0";
bgcolor = ["ivory"];
page_1 = new page
{
   body_label = new label
   {
      value = "This is a UFDL form.";
   }
}
```

Essentially, the form consists of one or more pages. A page contains zero or more items, like the label item in the example above. The items can be made from item types that are part of UFDL (labels, buttons, fields, automatic actions and so on), or from item types form designers create themselves. Pages and item types have certain default characteristics that form developers can modify by specifying various options.

Declarative Language

Statements in a UFDL form description are always maintained as being true, much as formula fields in a spreadsheet are maintained as true. The simplest example of this is a total field that adds up the contents of various dollar fields in a form. If one of the dollar fields changes, so does the total field.

What makes UFDL different from languages like C++ and Java in this respect is that the constant evaluation of dependencies is inherent in the language. A UFDL form requires no special procedures to be written in order to run evaluations; the evaluations run automatically whenever dependent data changes.

Extensible Syntax

UFDL was designed to be easily extensible for both form developers and the creators of UFDL.

- Form developers can create their own item and option types within forms (although currently they cannot set up inherited attributes for each type they create).
- The authors of UFDL can add new features to each new version of UFDL.

Open Protocol

UFDL is an open protocol. This gives developers the freedom to manipulate UFDL forms any way they want. Scripts can be written to dynamically create forms, modify forms, or extract specific information from forms. UFDL forms can themselves make requests to databases and populate themselves with the information returned. This flexibility allows developers to integrate UFDL forms into any application.

People with knowledge of C or C++ may wish to refer to Appendix D: UFDL for C and C++ Programmers. This appendix outlines UFDL's similarities to those languages.

2.2 Features of UFDL Forms

A UFDL form looks and behaves just the way you imagine an electronic form should. It can contain graphical elements, modifiable fields, and action items. You can organize a UFDL form into pages similar to the pages in a paper form and you can include navigational aids such as toolbars, tabbing instructions, and scroll bars. In addition, you can code the form to make logical decisions, to interface with other applications, and to automatically format and check user's entries.

A desktop form viewer application displays the forms. This UFDL form viewer allows users to enter input, enclose and view external files, and print and save forms. When it is convenient, the user can perform a simple action, such as pressing a button, to submit the completed form to an application for processing.

Some of the features that make UFDL forms ideal for every-day business use are outlined here.

Versatile Form Design

UFDL is very versatile. It provides many features you can use to customize both the appearance and functionality of your form.

Absolute and Relational Positioning Schemes

UFDL supports both an absolute positioning scheme and a relational positioning scheme. The absolute positioning scheme allows a form designer to place visible form items in fixed locations on a form. This is useful for beginners and for GUI design applications that

use a drag-and-drop method for designing forms. But an absolute positioning scheme is not a cross-platform solution. Used in conjunction with relational positioning, however, it can create modularized blocks of a form that can be easily moved around.

UFDL's relational positioning scheme allows designers to create forms that appear the same on any platform. It aligns visual elements in relation to other visual elements on the form, ensuring forms look consistent on all computers and at all screen resolutions. If an item changes size-either to accommodate a dynamically created value or a system font size-items aligned to it will shift in relation to it. This relational positioning scheme is flexible, giving developers freedom to create original layouts.

Support for User-Defined Objects

UFDL lets designers define their own form objects. These objects have no visible properties and initiate no actions, which means that form developers can store specialized information in the form without harming its appearance or behavior. A form viewer application respects references to custom objects in the form definition, allowing a custom object to accumulate information and also allowing other elements in the form to be altered according to the custom object's contents.

Input and Format Control

UFDL permits form designers to specify an item's availability, edit state, and input and output formats. This means the form can perform much of the data checking and formatting typically performed by form processing applications.

Digital Signatures

Version 4.0 and higher of UFDL supports digital signatures, for secure, tamper-proof documents. Digital signatures are incorporated into the description of the form, and allow the developer to specify that a user may sign the entire form or parts of the form. In addition, multiple users may sign a form.

Automatic Actions

UFDL supports automatic timed behavior activated by the form. Forms can automatically cancel themselves, submit themselves to a server for processing, open new forms, and upload information to a server.

The ability to perform automatic actions provides a mechanism that form designers can use to create stated connections with other applications. An application typically requiring a stated connection is a database management system.

Logical Operations and Arithmetic Computations

UFDL uses a set of options to describe a form object's appearance and behavior. For example, the option bgcolor describes an object's background color. UFDL permits form designers to use literal values or logically computed values (called computations) to determine the

value of an option.

These computations are resolved when the form appears. You can nest

computations, employ complex mathematical operations, populate and use arrays, and make decisions.

Computations provide designers with a very powerful and sophisticated tool for customizing forms to the needs of individual users and applications. It takes very little code (one line per logical computation) and it allows decisions regarding a form's appearance and behavior to occur at run-time.

Functions

UFDL functions allow forms to perform procedural logicas well as complex operations that would normally require complicated conditional statements. For details, see 7: UFDL Functions".

Stand Alone Definitions

All aspects of a form's appearance, behavior, and content are integral to the form definition. Therefore, unless you specify otherwise, the entire form definition and the user data travel with the form when a user submits it for processing. Consequently, you can transmit any UFDL form to any site with a UFDL-compliant form viewer application and the viewer will display the form correctly.

The only exception to this rule occurs when the form design specifies partial submission of forms. UFDL permits form designers to specify partial submissions in one of two ways:

* by specifying which parts to transmit

* by specifying HTML format

Partial submissions help reduce network traffic and transmission time.

Context Sensitive Help

UFDL provides a mechanism whereby form designers can define help messages for individual items in the form. Help messages appear in a window overlaying the form.

Enclosures

Users can enclose external files in UFDL forms. They can organize the files into folders, and they can display, copy, or remove the files. Enclosed files are encoded using the base64 encoding technique.

UFDL includes a MIME type with an enclosed file's description. This allows form viewer applications to choose an appropriate viewer (for example, World Wide Web browser, word processor, etc.) when displaying enclosures.

2.3 Description of a UFDL Form

A UFDL form is a collection of items (for example, buttons, labels, amd fields) organized into pages. There are items to display fixed values, items to collect user input, items to initiate actions, and items to assist with form navigation. The decision about which items to place on a page and how many pages to include in the form is application dependent.

UFDL provides a set of options for assigning characteristics to the form and to its pages and items. These include such things as the behavior, appearance, and location of an item. UFDL defines default settings for many of these options, or you can define your own settings in the form.

The following example describes a simple two-page form:

```
 version = "4.0.0";
 bgcolor = ["ivory"];

 page_1 = new page
 {
    bgcolor = ["seashell"];

    next_page_button = new button
    {
       value = "Next Page";
       url = ["#page_2.global"];
    }
 }
 page_2 = new page
 {
    fontinfo = ["Helvetica", "14", "plain"];

    hello_label = new label
    {
       value = "Hello, world.";
    }
 }
```

For information on the syntax rules of a form description, see "2.4-Syntax of UFDL"

2.3a What is a Page?

A form page is similar to a page in a paper form. Each page consists of its own set of items. You can place any number and type of items on a page. The number of items, their sizes, and their locations determine the size of the page.

> See the discussions of 'Relational and Absolute Positioning' and 'Item Placement' for more information on this topic.

In some senses, pages act like independent forms. They have their

own size, appearance, toolbars, and characteristics. As well, relational positioning of the page's items is based solely on other items on the same page.

The following example shows a page containing a label and a button:

```
page_1 = new page
{
   bgcolor = ["seashell"];


   hello_label = new label
   {
      value = "Hello, world.";
      fontcolor = ["blue"];
   }
   next_page_button = new button
   {
      value = "Next Page";
      url = ["#page_2.global"];
   }
}
```

For more information on the syntax rules of a page description, see '2.4-Syntax of UFDL'

Relational and Absolute Positioning

UFDL supports two positioning schemes for creating a page image: relational and absolute positioning. In the relational positioning scheme, each item's location depends on the location and size of one or more other items on the page. For example, a field might be below and slightly to the right of a label. A series of buttons might be placed to appear one after the other.

In the absolute positioning scheme, each visible item is anchored to a particular coordinate on the page drawn on the computer screen. Each coordinate represents a distance in pixels from the top left corner of the page. In addition, a form designer using absolute positioning can offset items from other items.

Absolute positioning is useful for graphic form design programs because it allows users to drag and drop items on a form. It is not a good cross-platform positioning scheme, although when used carefully in conjunction with relational positioning, it can be successful.

Relational positioning provides cross-platform compatibility in UFDL form designs, because all visible items are placed relative to each other. Therefore, if any item's size changes because of a change in font size or a dynamically generated value, other items on the form will shift to accommodate it, while maintaining their positions relative to each other.

For more information, see '2.4f-Item Placement'

Toolbars

The toolbar is a separate and fixed area at the top of a page.
It functions much like a toolbar in a word processing application.

Typically, you place items in the toolbar that you want users to see no matter what portion of the page they are viewing. Toolbars are optional and each page has its own toolbar.

The toolbar and the remainder (or body) of the page operate independently of one another. Both are scrollable, and scrolling one does not scroll the other. The toolbar can also have different characteristics than the page body, and relational positioning of toolbar items is based solely on other items on the same toolbar.

2.3b What is an Item?

Items are the basic elements of a page. Just as paper forms consist of items like lines, boxes, and instructions, UFDL forms consist of items like lines, boxes, text fields, labels, buttons, and so on. There are two categories of items:

- external

- internal or hidden

A page can include both categories of items.

> See the section 'UFDL Form Items' in section 4.0 for a description of each item.

External items occupy space on the page. They can be either visible or invisible. Visible items are things users see like labels and buttons. Invisible items are things like spacers that create white space on the form.

Internal items are invisible and occupy no space; instead they trigger form actions or store data used by other items. Action and data items are examples of internal items. An action item initiates a transmission, while a data item contains data stored in the form.

> An instance is a particular occurrence of an item type. For example, a form may have two labels. Each label is an instance of the item type 'label'.

Each type of item has default characteristics. For example, all fields will be a certain length and color unless the form developer specifies otherwise. A form developer can modify an item's default characteristics by adding options to its definition. For example, the field described below on the left would have a default appearance of 60 characters long and one row high (as well as having other default characteristics). On the right, the size option added to its description overrides that default size.

```
date_field = new field
```

```
    {
    }
```

```
 date_field = new field
 {
   size = ["20", "1"];
 }
 Field using default characteristics only
 Modified size overriding the default size
```

There are defaults for most item characteristics. If the defaults meet your requirements, an item definition may include only the instance identifier, a unique item tag. Instance identifiers are mandatory. They are critical to the relational positioning scheme. For that reason, UFDL incorporates the identifier into the syntax of an item definition.

An item's definition includes:

- An instance identifier (an item tag that uniquely identifies it).
- An open brace following the item declaration.
- A close brace at the end of the definition (after the options, if there are any).
- Optional information giving the item characteristics, including its position on the page, graphical characteristics and size, initial value and edit state, and instructions for handling the item when the form is submitted. Because these characteristics are optional, the lines that specify them are called options.

Here is a sample of an item description:

```
 date_field = new field
 {
  size = ["20", "1"];
  label = "Today's Date";
  format = ["date", "long"];
  value = "*";
  itemlocation = [["after", "name_field"]];
  }
```

For more information on the syntax rules of an item's description, see '2.4-Syntax of UFDL'

2.3c What is an Option?

See the section 'UFDL Form Options' in section 4.0 for a description of each option.

An option defines one characteristic of a form, a page, or an item. There are options to specify each aspect of the appearance and behavior of your form. Some options apply to the entire form, others apply only to items, and still others apply to pages or items. The example below shows options giving characteristics to an

entire form, to a page, and to a particular item.

```
 version = "3.2.0";
 bgcolor = ["ivory"];
```

```
page_1 = new page
{
...
page_1 = new page
{
bgcolor = ["seashell"];

bar_box = new box
{
...
bar_box = new box
{
bgcolor = ["black"];
size = ["60", "5"];
}
...
```

Options that appear at the top of the form, like the example on the far left, are called global settings. They apply to the whole form.

Options that appear at the top of a page, like the example in the center, are called page settings. They apply to the entire page. Page settings override any similar global settings-but only for the page on which they occur.

Options within items, like the example on the far right, apply only to the item whose description they are in.

2.3d Including External Files

See the '#include' section in section 2.8a for a description of the '#include' statement.

The UFDL #include statement allows you to include external files in your form definition much as you would include header files in a C language source file. The form viewer application replaces the #include statement with the contents of the file you specify. The included file must reside in a secure include directory accessible to the form viewer application.

2.3e Unrecognized Items and Options

User-Defined Items and Options and Newer UFDL Items and Options

As a UFDL form viewer parses a form, it ignores items and options it does not recognize. This feature has a number of advantages.

* It allows a form designer to include items and options for new form viewer applications without affecting the form's behavior in other viewers.

* Form processing applications can use the custom items and options when processing the form. One example of a custom item might be an SQL query item the application uses to populate a

response form.

Unrecognized items and options include:

* User defined (or custom) items and options.

* Items and options from releases of UFDL that are newer than the user's form viewer application understands.

2.4 Syntax of UFDL

2.4a Basic Syntax Rules

The basic syntax rules of UFDL are:

* It is case sensitive.

* It ignores white space around and within statements.

* It permits multiple line statements.

* It permits multiple statements per line.

2.4b Form Definition

The syntax of a UFDL form definition is as follows:

```
<version definition statement>*
<option definitions for the form characteristics>
<page definition1>
...
<page definitionn>**
```

* mandatory statement. See 'version' on page 226 for the syntax of this statement.
** there is no limit placed on the number of page definitions in a form; however, every form must contain at least one page definition.

For example,

```
version = "3.2.0";
bgcolor = ["ivory"];
fontinfo = ["Helvetica", "10", "plain"];

//This is page 1
page_1 = new page
{
   <option definitions for the page settings>
   <item definitions for items located on page 1>
}
```

```
    //This is page 2
    page_2 = new page
```

```
     {
        <option definitions for the page settings>
        <item definitions for items located on page 2>
     }
     ...

     //This is page 10
     page_10 = new page
     {
        <option definitions for the page settings>
        <item definitions for items located on page 10>
     }
```

Defining global settings for the form is optional. It has the effect of setting characteristics that apply to the entire form. In the previous example, version, bgcolor, and fontcolor are global settings. These characteristics override the defaults defined by UFDL. Specific pages and items will override these global settings if the same option has been defined differently for that page or item.

2.4c Page Definition

The syntax of a page definition is as follows:

```
 <page tag> = new page
 {
    <option definitions for the page characteristics>
    <item definition1>
    ...
    <item definitionn>
 }
```

Notes:

i) The braces are mandatory.

ii) A page definition must begin on a new line.

iii) Item definitions are optional and there is no limit placed on the number of item definitions in a page.

The page tag uniquely identifies a page instance. No two page tags in a form can be the same. See the section 'Identifiers' on page 39 for tag naming conventions.

Defining page characteristics is optional. It has the effect of setting options that are global to that page. These characteristics override the defaults defined by UFDL and any global options set by the form characteristics. Specific items will override the page settings if the same option has been defined differently for that item.

In the following example, you can see a sample page definition. The page tag is Page_one and the page contains a label and a

button. The page has a background color of cornsilk and each item on the page will have a font of Times 14.

```
 Page_one = new page
 {
```

```
      bgcolor = ["cornsilk"];
      fontinfo = ["Times", "14", "plain"];
      button_label = new label
      {
        <option definitions for the label characteristics>
      }
      save_button = new button
      {
        <option definitions for the button characteristics>
      }
     }
------
```

2.4d Item Definition

The syntax of an item definition is as follows:

```
 <item tag> = new <item type>
 {
  <option definition1>
  ...
  <option definitionn>
 }
```

Notes:

i) The braces are mandatory.
ii) An item definition must begin on a new line.
iii) Option definitions are optional.
iv) You cannot assign values to options in other item definitions.

The item tag uniquely identifies an item instance. No two item tags on a page can be the same. See the section '2.5a-Identifiers' for tag naming conventions.

Item type is a name that identifies the type of item. Examples of item types are: button, label, field, line, and check. See the section 'UFDL Form Items' on page 58 for a description of each item type.

> Tip: You can also define and use your own item types and options. See the '<custom>' item and option descriptions later in this manual.

There is a finite list of UFDL-defined options applicable to each type of item. You can code as many or as few from the list as you wish. There are default settings for most options (defined by UFDL). You may choose to use those defaults or to define your own settings. Defining your own settings overrides the defaults.

You can also create your own item types. A UFDL parser will ignore these custom item types, but you can use them to store information specific to your application, and then refer to them in other item descriptions in the form. For more information on how to refer to

options in the form, see 'Referring to Other Options' later in this section.

In the following example, you can see a sample button definition.

The button has the following characteristics:

 * The item tag is save_button.

 * It will save the form to a file on the user's workstation.

 * The button's label is Save Form.

 * The background color is cyan.

 * The font used for the label is Helvetica 12.

```
save_button = new button
{
 bgcolor = ["cyan"];
 fontinfo = ["Helvetica", "12"];
 type = "save";
 value = "Save Form";
}
```

2.4e Item Size

Every external item has a characteristic shape. Many items also contain data such as text and images. This is the basic item. For example, the basic field is a rectangular space where users can input text. Buttons are rectangular objects containing a descriptive label.

Items may also contain the following elements:

 * borders

 * an external label

 * scroll bars

Borders are lines outlining an item's shape. Their use is optional and their thickness is variable.

External labels are part of an item's definition but they occupy their own space. An example of an external label is the label you define for a field. This label occupies space above the field item.

Several types of items permit users to scroll the data the item contains. Typically, scroll bars appear with these items. Examples of items permitting scrolling are fields and lists.

Size Calculation

There are two sizes calculated for an item. They are:

 * basic item size

* the item's bounding box

The basic item size is composed of the item's characteristic shape

and any imbedded data. UFDL defines a set of default basic item sizes. You can choose to use these defaults or you can define the size using the size option. When deciding whether to define the size and what size to specify, you will want to consider any data imbedded in the item.

The bounding box is an unseen rectangular area surrounding each item and including all elements of the item. The size of the bounding box depends on the sizes of the various elements. UFDL calculates this size, taking into account the basic item size and the existence and size of the various optional elements. For example, if the item definition contains a borderwidth setting (meaning the item has a border), then the bounding box size encompasses the basic item and the space occupied by the border.

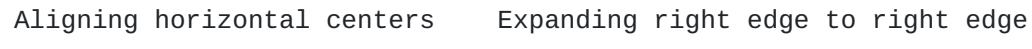
See 'Appendix B: Default Sizes' for the default item and bounding box sizes.

Altering Size Dynamically

You can dynamically alter the bounding box size, and thus the basic item size and the space available for the external label. The itemlocation option contains various directives permitting you to do this.

> An item's vertical center is halfway between the top and bottom edges. The horizontal center is halfway between the left and right edges.

A bounding box has six edges: left, right, top, bottom, vertical center, and horizontal center. You can align any of these edges with the edge of another item's bounding box (called a reference item in this context). Once you have aligned one edge, you can expand the bounding box until the far edge aligns with another location. In this manner, you override the bounding box length in that direction.



For example, you can align the left edge with the horizontal center of one reference item. You can then expand the right edge until it aligns with the right edge of the original reference item or a second reference item. This pair of directives sets the bounding box width.

2.4f Item Placement

UFDL supports two different positioning schemes to place external items on a page: relational positioning and absolute positioning.

Relational positioning means an item's location depends on the location and size of one or more other items on the page. This feature is similar to the mechanism used for dynamic sizing.

Relational positioning uses the bounding boxes of the other items as reference points. Items align relative to these bounding boxes. You must define the location of the other items before you can use them as reference points.

The itemlocation option provides various directives you can use to specify an item's location. For example, you might place an image before a radio and expand its bottom edge to the bottom edge of the radio button.

Positioning the image before radio buttons
Expanding the bottom of the image to the bottom of the radio buttons.

The only items whose placement is not affected by relational positioning are the first item in the toolbar and the first item in the body of the page. The first item assigned to the toolbar goes in the top left corner of the toolbar. The first item not assigned to the toolbar goes in the top left corner of the body.

Absolute positioning places an item in an absolute position on the page, anchoring it to a particular coordinate. This coordinate is a pair of pixel measurements defining the item's distance from the top left corner of the page.

Absolute positioning also allows items to be offset from their original position, in order to make layout with an absolute positioning scheme more flexible. When offsetting an item, the form developer first places the item on the page and then specifies how far it should be offset from that position.

The absolute positioning scheme's advantage is that it makes designing a drag-and-drop form designer easy. Absolute positioning is not a good cross-platform solution, however, and in order to ensure that forms appear consistent on all platforms, developers should use either strictly the relational positioning scheme, or a careful combination of relational and absolute positioning.

For more information, see the itemlocation option description in section 5.19.

2.4g Toolbar Definition

A toolbar is a section that stretches across the top of a page in which items can be placed for quick access. If a user scrolls down on a page, the toolbar remains visible.

A user defines a toolbar using the toolbar item. Each page can have one toolbar, and the toolbar will appear on only that page. Place items in the toolbar by using the within modifier of the

itemlocation option.

The following example shows the definition of a toolbar with two

```
    items: a label and a close button.
     p1_toolbar = new toolbar
     {
      bgcolor = ["cyan"];
     }
     title_label = new label
     {
      value = "Student Registration Form";
      fontinfo = ["Helvetica", "16", "bold"];
      itemlocation = [["within", "p1_toolbar"]];
     }
     close_button = new button
     {
      type = "close";
      value = "Close Form";
      itemlocation = [["within", "p1_toolbar"], ["below", "title_label"],
                           ["alignhorizc2c", "title_label"]];
     }
------
```

2.4h Option Definition

An option definition is an assignment statement that assigns one characteristic to an item, a page, or to the whole form. The expression on the right hand side of the equal sign contains the option's setting. The syntax of an option definition statement is as follows:

```
 <option identifier> = <expression>;
 Note: The semicolon is mandatory and terminates the statement.
```

```
For example:
 value = "Submit Form";
 fontinfo = ["Helvetica", 16", "bold"];
 url = global.global.db_address
```

Explanation of Syntax

Option identifier is a name that identifies the type of option. It can be a UFDL-defined option or a user-defined option. Examples of option identifier are: bgcolor, fontinfo, itemlocation, and size. See the section 'UFDL Form Options' in section 5 for a description of each option and its possible values.

An expression specifies a value. An expression can be any of the following:

```
 * a literal
    (for example, the right hand side of
     value = "Submit Form"; )
 * a reference to another option definition in the form
    (for example, the right hand side of
```

```
         url = global.global.db_address; )
     * an operation
        (for example, the right hand side of
         value = total_field.value + "3400"; )
```

```
     * an array specification
        (for example, the right hand side of
         fontinfo = ["Helvetica", "16", "bold"]; )
------
```

2.4i Literals

Specify a literal as a quoted string. This is true even for operands of an operation. Examples of using literals are:

```
   "V3.2.0" - yields "V3.2.0"
   "1" + "2"- yields "3"
   "UFDL\\" +. "form1"- yields "UFDL\\form1"
------
```

2.4j References to Other Options

In order to copy information from one place in the form to another, or to make a decision based on the contents of items in the form, a developer needs to refer to one or more other options in the form.

This is done using an option reference. The referenced option definition can exist anywhere in the form definition, including after the current statement.

For examples of option references, see the paragraphs following the box below.

An option reference has several possible formats:

1. for options in the current item definition use one of the following:*
   * <option reference>
   * <option reference>-><option reference>
2. for options in another item definition use one of the following:*
   * <item reference>.<option reference>
   * <item reference>.<option reference>-><option reference>
3. for options in page characteristics use one of the following:
   * global.<option reference>
     - for characteristics on the current page
   * <page tag>.global.<item reference>
     - for characteristics on another page
4. for options in form characteristics use:
     global.global.<option reference>

   where <option reference> is one of:
   * <option identifier>
     - for the complete option setting (it can be a single value or an array)
   * <option identifier>[<array element>]
     - for one element of an array**

   and <item reference> is one of:
   * <item tag>

- for items on the current page
* <page tag>.<item tag>
- for items on another page

and the indirect membership operator (->) indicates:

* a dynamic option reference

* The phrase '-> <option reference>' can occur any number of times on the right hand side of an option statement.
** See 'Array References' in section 2.4n for information on <array element>.

In order to refer to an option that varies depending on what the user of a form enters, use a dynamic option reference. For example, a form developer cannot know what cell a user will choose in a popup menu. To refer to the value of whatever cell the user chooses, the developer must use a dynamic option reference. For example:

```
popup_menu.value->value
```

A dynamic option reference (-> <option reference>) provides a mechanism for determining the location of the option at run-time. The references preceding the indirect membership operator must resolve to an item reference or a reference to the form or page characteristics.

Examples of option references are:

- an item on the current page:
  list_one.value
  This identifies the value option of the item whose item reference is list_one.

- a form characteristic:
  global.global.bgcolor
  This identifies the bgcolor option specified at the top of the form, as a form global characteristic.

- a dynamic option reference with one level of indirection:
  The my_choice.value setting becomes the item reference for the bgcolor option. If, for example, my_choice.value contains "global", then this reference is equivalent to global.bgcolor.

- a dynamic option reference with two levels of indirection:
  my_choice.value->value->bgcolor
  The my_choice.value setting becomes the item reference for the value option. If, for example, my_choice.value contains "your_choice", then my_choice.value->value is equivalent to your_choice.value.

If your_choice.value contains "page_two.global", then the complete reference is equivalent to page_two.global.bgcolor.

2.4k Relative Page Tags and Item Tags

UFDL contains a way to refer to pages and items without using specific identifiers: relative page tags and item tags. For example:

```
  value = itemnext->value;

  itemlocation = [["after", itemprevious]];

  url = ["#"+.global.pagenext->global];

  url = [global.global->pagefirst->itemfirst->url[0]];
```

Refering to items and pages that don't exist yet

Relative page tags and item tags are particularly useful if you are making template forms for an application that dynamically generates extra items and pages during run time.

Since dynamically-generated items and pages don't exist until runtime, you cannot refer to them by name when you are coding the template form (since you don't necessarily know what name the generation program will use). Relative page tags and item tags allow you to refer to non-existent pages and items.

For example, you might want to add a paging button that opens the next page of a form when the user clicks it. Normally, if your next page was called page_2, you'd set up the paging button's url to:

```
 url = ["#page_2.global"];
```

But if the next page will be dynamically generated by a program, you don't know what page tag to put in the url. So you would use a relative page tag, like this:

```
 url = ["#"+.global.pagenext->global];
```

Available relative page tags and item tags

pagefirst

Meaning First page in form description.
Reference must start at Form global (global.global).
Example

```
url = [global.global->pagefirst->submitButton.url[0]];
```

pagelast

Meaning Last page in form description.
Reference must start at Form global (global.global).
Example

```
value = global.global.pagelast->resultField.value;
```

pageprevious

Meaning Previous page in form description.
Reference must start at Page global (global).
Example

```
url = ["#"+.global.pageprevious->global];
```

pagenext

Meaning Next page in form description. The last page points to the first page.

```
      Reference must start at Page global (global).
      Example
      url = ["#"+.global.pagenext->global];

    itemfirst

      Meaning First item in page description.
      Reference must start at Page global (global).
      Example
      value = global.itemfirst->value;

    itemlast

      Meaning Last item in page description.
      Reference must start at Page global (global).
      Example
      value = global.itemlast->value;

    itemprevious

      Meaning Previous item in page description. First item points to
       last item in page description.
      Reference must start at Item level.
      itemlocation = [["after", itemprevious]];

    itemnext

      Meaning Next item in page description. Last item points to first
       item in page description.
      Reference must start at Item level.
      Example
      value = itemnext->value;

   Rules for creating references using relative tags

    To create references using relative page tags and item tags:

     1. Follow the normal rules for page, item, and option references
        (see "References to Other Options" earlier in this section),
        except apply the rule in 2, below.

     2. Use the dereference symbol ( -> ) following the relative tag,
        if the tag is followed by another tag. For example:
         itemlocation = [["after", itemprevious]];
        but
         value = itemprevious->value;
        and
         value = global.global.pagefirst->itemfirst->value;

    If you do not use the dereference symbol, a UFDL parser will
    evaluate the relative tag as an option name.
------
```

An operation is a calculation or a decision. The syntax of a calculation is one of the following:

1. <operand> <math operator> <operand>
2. <operand>

Operands that are numbers can have a unary minus. An operand can be any of the following:

- a literal (for example, "3")
- a reference to another option (for example, total_field.value)
- a calculation (for example, "total_field.value" *"4")
- (<decision>) - see below for the syntax of a decision

A math operator can be any of the following:

- additive operator
- multiplicative operator
- exponentiation operator

See the table of operators below.

The syntax of a decision is as follows:

<comparison> ? <expression> : <expression>
where <comparison> is:

- <Boolean> <logical operator> <Boolean>

and <Boolean> is:

- <operand> <relational operator> <operand>

Note: See the table below for the definition of logical and relational operators.

In decisions:

* An operand can have a logical NOT (!) before it.
* An expression cannot be an array.

Some examples of decisions are:

* A decision based on a check box.
  male_check == "on" ? "male" : "female"

If the check box is selected, or on, then the result will be male. Otherwise, the result will be female. This decision could be used to set an item's value.

* A decision based on a value.
  name_field == "Smith" ? "on" : "off"

If the name entered into the name field is Smith, then the result will be on. Otherwise, the result will be off. This decision could be used to set an item's active status.

UFDL recognizes the following operators:

Type of Operator	Symbol	Operation
Additive	+	addition
	-(minus)	subtraction

	+.	concatenation
Multiplicative	*	multiplication
	/	division
Exponentation	^	exponential

Relational	>	greater than
	<	less than
	<=	less than or equal to
	>=	greater than or equal to
	==	equal to
	!=	not equal to
Logical	&&	AND
	||	OR
	!!	NOT
Unary Minus	-(minus)	take negative
Decision	x?y:z	Assign the value of expression y to the result if expression x evaluates to true. Otherwise, assign the value of expression z to the result.
Assignment	=	Assign right operand to left operand
Membership	.(dot)	structure membership
	[]	array membership
	->	indirect membership

Precedence of Operations

Operations are evaluated in the following order:

- membership
- exponentiation*
- multiplicative and unary minus*

- additive
- relational
- logical NOT
- logical AND
- logical OR
- conditional

Operations at the same level of precedence are evaluated from left to right.
Parentheses override the precedence levels; however, operations within parentheses are evaluated using the normal precedence levels.

- When a unary minus immediately follows an exponentiation symbol (^), the unary minus is evaluated first. For example, 10^-5 is evaluated as ten-to-the-minus-five.

Concatenation

An addition operation may imply concatenation. If either operand in the addition contains a non-numeric value, the operands are concatenated. Otherwise they are added arithmetically. You only need to use the concatenation operator if both operands are, or can be, numeric values.

The following examples demonstrate this rule:
"UFDL\\" + "form1" -yields "UFDL\\form1"
"UFDL\\form" + "1" - yields "UFDL\\form1"
"1" + "2" -yields "3"
"1" +. "2" -yields "12"

The last example would not have resulted in concatenation without using the concatenation operator.

Separators

There are two separators in UFDL: comma (,) and semicolon (;). The comma separates list entries; the semicolon terminates an option definition statement.

2.4m User Events and Changes of State

An "event" is the user's act of causing the state of something in the form to change. For example, when the user clicks a check box, its state changes from being unchecked (or off), to being checked (or on). This act of causing the state to change from off to on is the event.

Other examples of events are:

- The user moves the mouse pointer over a button and its state changes from not having the mouse pointer over it to having the mouse pointer over it.
- The user switches from the first page to the second page in a form, and the state of the first page changes from being active to being no longer active, while the state of the second page changes from not being active to being active.

Recording Changes of State

All states in a UFDL form are recorded in options.

For example, in check boxes the state of "being-checked or not" is recorded in the value option. When a user checks a check box, its value changes from off to on. For all visible items, all pages, and for the entire form, the state of "having the mouse pointer over me or not" is recorded in the mouseover option.

This provides a form developer with enormous potential for creating intelligent forms that set and change themselves dynamically, based on changes of state. Those changes can be changes of user input (such as checking a check box), or simply changes of user behavior (such as moving the mouse pointer over a button).

Example

The following example illustrates using an event (a change of state caused by the user) to trigger self-modifying behavior in the form:

```
     saveButton = new button
        {
         type = "save";
         value = "Save";
```

```
         bgcolor = [mouseover=="on" ? "white" : "gray90"];
        }
```
The bgcolor option in the button above will change from gray to white when the user moves the mouse pointer over the button.

2.4n Arrays

UFDL uses arrays to store values in options requiring multiple settings. The number of elements and the number of dimensions in an array depend on the option. However, the syntax of the language supports n-elements and n-dimensions. Moreover, UFDL supports arrays containing a mix of simple elements and sub-arrays.

The syntax of an array is as follows:
 [<element1>, <element2>, ... <elementn>]
 Note: 'n' is the number of settings in the option.

An element can be either of the following:
- an expression
- an element definition statement

Element Definition Statements

The element definition statement allows you to assign a variable name to an element. Variable names permit you to refer to the element by name rather than by its position in the array. The syntax of an element definition statement is:
 <variable> = <expression>

UFDL syntax includes variable names in some arrays. In this case, you must use an element definition statement when assigning values to the element. For example, the format option syntax specifies names for the check option's range, length and template. To assign values to any of these elements, you must use the name specified in the syntax.

Examples of assignment statements using element definition statements:

```
 using a UFDL-defined variable name
    format = ["integer", range=["1","100"]];
 using a user-defined variable name
    delay = [the_repeat = "once", the_time = "10"];
 using both UFDL-defined and user-defined variable names
    format = [the_type = "string", length = ["5", "25"]];
```
Array elements for UFDL-defined option types must be coded in the position they are documented in this specification, unless they have UFDL-defined variable names listed in this specification. For example, the elements for the size option must always be in the order [width, height], but the elements for check and format types in the format option can be in any order, since they have UFDL-defined variable names.

Decisions in Arrays

A decision can be used to determine any element within an array, so

long as that element is not itself an array.

For example, the following format line is valid.
 format = [check_1.value == "on" ? "string" : "integer"];

The decision sets the data type to be a string if check_1 is "on",
or an integer if it is "off".

This format line is not valid:
 format = ["integer", check_1 == "on" ? range = ["10","20"] :
 range = ["0","10"];

The range is itself an array, so a decision cannot be used to
determine which range should apply.

Array References
 The syntax of an array reference is:
  <array name>[<array element>]
  Note: Repeat the phrase '[<array element>]' until reaching the
        desired depth.

 Array name is an option or variable identifier. Array element can
 be one of two things:
     - a number indicating the position of the element in the array
     - variable name

 Before using a variable name, you must define the name in an
 element definition statement. See the section 'Element Definition
 Statements' in section 2.4n for more information.

 UFDL array starting position is 0; therefore, a reference to the
 first element of the array is really a reference to element zero
 (0).

 The following examples show various array assignments and
 references:

  itemlocation = [["below", "field1"], ["alignl2l", "field2"],
                 ["alignr2c", "field3"]];
     itemlocation[0][1]   - points to "field1"
     itemlocation[2][0]   - points to "alignr2c"
  itemlocation = [the_pos= ["below", "field1"], the_align=
                 ["alignl2l", "field2"]];
     itemlocation[the_pos][1]   - points to "field1"
     itemlocation[the_align][0]   - points to "alignl2l"
  format = ["integer", range=["1","10"]];
     format[0]   - points to "integer"
     format[range]   - points to ["1", "10"]
  format = ["integer", range=[the_low="1", the_high="10"]];
     format[range][the_low]   - points to "1"

2.4o Defining Tabbing and Paging

UFDL provides two mechanisms for defining the movement between pages and items in a form.

- tabbing to the new item or page
- linking to the new page

You can combine these methods or you can choose to use only one.

The item in focus is the item with the cursor and, often, with some form of highlighting.

Tabbing permits the user to move from one item to another and from one page to another using a keystroke. Linking permits the user to select a form item whose action moves the focus to a new page.

Note: The only items users may tab or link to are modifiable items. These are items users can change or select.

Tabbing

To use tabbing, define a tabbing sequence using the next option. The sequence can include items anywhere in the form. Define the tabbing sequence this way:

- Define the first item in the sequence by including the next option in the form characteristics. When the form opens, the page containing this item displays with the item that is in focus.
- Define each subsequent item by including the next option in the definition of each item in the sequence.

The next option setting is the item reference of the next item to receive focus (that is, the referenced item). When the user tabs from the current item, the referenced item receives the focus. If the item is on a different page, the current page closes and the new page displays.

You can use tabbing to display a new page without choosing an item to activate. Set the next option to the characteristics reference for the new page. This displays the new page and focusses on the first item in that page's tabbing sequence. The page characteristics reference is <page tag>.global.

- Define the first item in a page's tabbing sequence by including the next option in the page characteristics.

This example shows a simple tabbing sequence:

```
version = "4.0.0";
// Open the form on page 'page_one' and focus on 'title_list'.
next = "page_one.title_list";
page_one = new page
{
 // Define the default first item in this page's tabbing sequence.
 next = "name_field";
 bgcolor = ["LightBlue"];
 form_title = new label
```

```
 .......
}
title_list = new list
{
```

```
   // Tab to the 'name_field' item from here.
   next = "name_field";
   ......
  }
  name_field = new field
  {
   // Tab to the 'your_signature' item on 'page_two' from here.
   next = "page_two.your_signature";
   ......
  }
 }

 page_two = new page
 {
  bgcolor = ["PaleGreen"];
  form_title = new label
  {
   ......
  }
  your_signature = new tablet
  {
   // Tab back to page 'page_one' and focus on the first item
   // in the page_one tabbing sequence.
   next = "page_one.global";
   ......
  }
 }
```

Linking

To use linking, define action, button, or cell items for the links you want to include in the form, and set their type option to pagedone. Since each item performs only one link, you require a separate action, button, or cell for each link. This method is often best suited to defining links to new pages.

When you link to a new page, do one of the following:

- Specify an item on the new page for the focus to move to.
- Specify that the focus move to the default position on the new page, by "linking" to the page characteristics section.

Store the reference of the linked item or page in the url option of the action, button, or cell. The reference is an item reference or a page characteristics reference. Use an item reference when you want to link a specific item. Use the page characteristics reference when you want to link the first item in the page's tabbing sequence. A page characteristics reference is <page tag> .global.

When the link occurs (i.e., a user selects the button), the current page closes and the linked page appears. Before the current page can close, all fields containing error checking must be correctly

filled in.

This example shows how you might use linking:

```
 version = "4.0.0";
```

```
  // Open the form on page 'page_one'. Allow the first item in the page's
  // tabbing sequence to receive focus.
  next = "page_one.global";
  page_one = new page
  {
   // Define the default first item in this page's tabbing sequence.
   next = "name_field";
   bgcolor = ["LightBlue"];
   form_title = new label
   {
    .......
   }
   name_field = new field
   {
    ......
   next_page = new button
   {
    value = "Page 2";
    // Link to the next page. Allow the first item in the page's
    // tabbing sequence to receive focus.
    type = "pagedone";
    url = "#page_two.global";
    ......
   }
  }

  page_two = new page
  {
   // Define the default first item in this page's tabbing sequence.
   next = "your_signature";
   bgcolor = ["PaleGreen"];
   form_title = new label
   {
    ......
   }
   your_signature = new tablet
   {
    ......
   }
   first_page = new button
   {
    value = "Page 1";
    // Link to the first page. Allow the first item in the page's
    // tabbing sequence to receive focus.
    type = "pagedone";
    url = "#page_one.global";
    ......
   }
  }
```

UFDL-Defined Default Paging and Tabbing Sequence

UFDL-defined default sequence depends on the order in which you define pages and items in the form. The default first page is the first page defined in the form. The default first item is the first

item defined for the body of that page.

The sequence progresses through the page definition moving from one modifiable item to the next. If a user tabs past the last modifiable item on the page, focus returns to the first modifiable item in the page's toolbar (if one exists) or the first modifiable item on the page. The default sequence does not permit you to move between pages.

UFDL permits you to define pages and items in any order, regardless of when and where they display. If you define your pages and items in a random order, the default sequence may result in apparently random movement.

2.4p Including External Files

You can code a #include statement anywhere in a form definition except imbedded in another statement. You can also nest #include statements. See the section 'UFDL Form Viewer Directive' in section 6.1 for a syntax of the #include statement.

In the following examples, you can see the #include statement used in a variety of locations.

```
// Use the standard defaults for v3.2.0 forms. This include file
// contains the 'version' option statement and the default 'url'
// option statement.
#include "v3form.txt"

page_one = new page
{
 // Page one must contain the company logo. This include file
 // contains the 'label' and 'data' item definitions.
 #include  "co_logo.txt"

 // The remaining items are specific to this form.
 ...
}

// The last page is standard for all company forms. Use the
#include "lst_page.txt"
}
```

2.5 UFDL Language Elements

2.5a Identifiers

Identifiers are the names you assign to the following entities:

- page tags
- item tags
- option names

- variable names
- datagroup names
- group names

The naming conventions for an identifier are as follows:

- It must begin with an alphabetic character.
- It can contain any of the characters A-Z, a-z, 0-9, $ and underscore.

An example of a valid identifier is sql_query.

2.5b Custom Item Types and Custom Option Names

These are the names you assign to your own items and options. The naming conventions for a custom name are as follows:

- It must begin with an alphabetic character.
- It can contain any of the characters A-Z, a-z, 0-9, $ and underscore.
- It must contain an underscore.

2.5c Reserved Words

UFDL reserves the following words for its own use:

- UFDL item, option and variable names
- global
- page
- new

2.5d Quoted Strings

The syntax of a quoted string is:

"<character string>"

The minimum length of the string is one (1) byte; the maximum length is the lesser of two gigabytes (231 - 1 bytes) and the amount of memory the system will allocate.

Long quoted strings can span multiple lines. To code a multiple line string, break the string into segments and surround each segment with quotation marks. A reasonable segment length might be the maximum line length permitted in your text editor.

The following example shows a multiple line quoted string in an assignment statement. UFDL treats the segments as contiguous, ignoring any white space between them.

```
value = "This example demonstrates the use of quoted strings "
        "that span multiple lines.";
```

Some characters, such as tabs and line delimiters, are invalid in a quoted string unless you use an escape sequence. All escape sequences begin with the escape character (\). The following table shows the escape sequences UFDL recognizes and the characters they represent.

Escape Sequence	Character	Comments

\t	tab	UFDL interprets this as an imbedded tab character.
\n	line delimiter	UFDL interprets this as am imbedded line delimiter.

\xnn	hexadecimal number	UFDL interprets 'nn' as a hexadecimal number.
\mnn	octal number	If 'm' is 0,1,2,or 3, UFDL interprets 'mnn' as an octal number
\"	double quote	UFDL interprets this as an imbedded double quote mark.
\\	backslash	UFDL interprets this as an imbedded
\<any other>	<any other>	UFDL ignores the escape character.

2.5e Binary Data

See 'mimedata' in section 5.27 for the syntax of the assignment statement.

Images and sounds are examples of binary data. Store binary data in a form using the mimedata option of a data item. The mimedata option requires a quoted string as its setting.

To store binary data in this manner, you must first convert it to base64 format, copy the converted data into the form definition, and insert the quotation marks. In all likelihood the data will span several lines. Enclose each line in quotation marks.

Converting binary data to base64 format ensures the string contains no characters requiring an escape sequence.

2.5f Comments

Comments must occur at the end of the line or on a line by themselves. UFDL supports two comment formats:

```
// comment          - the comment ends at the end of the line
/* comment */       - these comments can span several lines
```

2.6 Security

Version 4.0 and higher of UFDL supports digital signatures, for secure, tamper-proof documents. Digital signatures are incorporated into the description of the form, and allow the developer to specify that a user may sign the entire form or parts of the form. In addition, multiple users may sign a form.

Design Goals Behind Digital Signatures in UFDL

Standards Based Security: Due to the sensitive nature of the issues surrounding data security and integrity, all digital signature technologies used in UFDL must be based on commonly accepted industry standards.

Vendor Independence: Any API used must have the ability to be

seamlessly replaced with another, should the situation warrant it.

Optional Implementation: The act of digitally signing a UFDL form

should not alter it in any way that prevents it from being opened by a viewer that does not support the digital signature technology.

Partial Content Protection: Forms must be able to be signed both in whole and in part to allow for sections to be approved by different signing authorities.

Incremental Protection: Forms or form sections must be able to be signed several times to allow for layered or incremental authorization.

How Digital Signatures Work in UFDL

User-level digital signature functionality is accessible through one or more signature buttons, which the form developer must place in the form. Typically, a signature button that is associated with a valid digital signature will display the signer's identity. A signature button associated with an invalid signature will display the word Invalid. Finally, a signature button that is not associated with a signature will appear to be empty. The form developer can override this basic behavior.

Subject to some constraints specified by the form, a user can access the following digital signature functionality: verify the digital signatures present in the form, view the signatures in a form, digitally sign all or part of a form, and delete the user's signature.

Signature Verification: The signatures in a form will be verified automatically when the form is first displayed. The user will be warned if any signature verification fails (in addition to the invalid appearance displayed by any associated signature buttons).

Signature Viewing: By pressing a signature button on the form, the user will be able to call upon a Signature Viewer dialog box to present the signature associated with the button. The dialog will include four buttons: OK, Sign, Delete, and Advanced.

The default button is OK, so the user can hit the Enter key at any time to release the Signature Viewer dialog. If there is no signature associated with the signature button, then the Delete and Advanced buttons will be inactive, and text field will be empty. The Sign button will be active, and it will receive the focus when the Signature Viewer dialog is first displayed so that the user can simply hit the space bar to sign the form. If there is a signature associated with the signature button, then the Sign button will be inactive, the Delete button may or may not be active, and the Advanced button will be active. The Advanced button will also receive the focus so that the user can hit space to view the advanced information regarding the signature.

The dialog will also use labels to display the signature status (No Signature, Signature Is Valid, Signature Is Invalid), the signer's identity (e.g., John Doe Manager, jdoe@company.com), the identity of the cryptographic service provider used to generate and verify the signature (e.g., Microsoft Base Cryptographic Provider

v.1.0), and the hash algorithm used (sha1 or md5).

Finally, the dialog will provide a readonly text field labeled Certificate Chain that will contain a textual description of the chain of certificate issuance for the signer. If there is no signature, then all elements will be empty except the signature status. If there is a signature, and the user presses the Advanced button, then a second dialog box will appear, containing a readonly text field and an OK button. The OK button will release the advanced dialog if pushed, and the text field will contain a UFDL text representation showing what the user signed.

Digital Signature: If the user presses the Sign button, then the viewer will first obtain the digital signature key of the user. If the user has more than one digital signature identity, then a dialog will be presented allowing the user to select the desired identity. The signature will then be created in accordance with the specifications of the signature button, and the signature demographics will be placed into the labels and fields of the Signature Viewer dialog. The Sign button will be grayed out, and the Delete and Advanced buttons will be activated. The Advanced button will receive the focus. Thus, the user may then press the space bar to view the new signature's advanced information (or Enter to release the signature dialog via the default OK button).

Deleting a Signature: If a valid signature is associated with the signature button, and it belongs to the current user, and the deletion of the signature will not corrupt other signatures on the form, then the Delete button will be activated. The user is not permitted to delete an invalid signature (even if it claims to have the user's identity because it may be lying). If the user presses this button, then the Signature Demographics field will be emptied, the Delete and Advanced buttons will be deactivated, and the Sign button will be activated and focused. Thus, the user can hit Enter to release the Signature Viewer dialog box via the OK button, or space to sign the form. Typically, the user will delete a signature, release the dialog, make some changes to the form, then reinstantiate the dialog to sign the changed form.

Freezing Option References, Calculations, and Other Formulas

Once an option has been digitally signed, it maintains the signed literal value and will not change, even if the option setting is a formula (for example, value=field1.value).

The literal value is stored in a start value element of the option name, which is represented by an open angle bracket with the value in quotation marks and a close angle bracket on the left-hand side of the equal sign, like this:

```
value<"Jane E. Smith"> = page1.nameField.value;
```

The viewer sets this literal value when a form is signed, submitted, or saved (and discards any old value if necessary). Because a digitally signed formula never fires after being signed, the start value for the option is always the same-and therefore it

is possible to reference the option and get the signed literal value.

Non-Compliance

In accordance with the design criterion of optional implementation, viewers that don't support digital signature should, for the most part, ignore the digital signature enhancements in a form. Viewers that support any UFDL versions prior to v.4.0 not be able to parse v.4.0 forms, because of the inclusion of the start value syntax starting in v.4.0 (see "Freezing Option References, Calculations, and Other Formulas" above). However, if a UFDL viewer v.4.0 or higher does not have digital signature capability, then the viewer should not attempt to verify the signatures. Further, if a button of type signature is pressed, the user should be informed that the digital signature feature is not supported.

If a digital signature fails because the signer's certificate is out of date, the viewer will denote this as a failure even if everything else checks out. The principal idea behind certificate expiry is that keys involved in a signature cannot be trusted beyond the expiry date.

Setting Up a Digital Signature Button

To allow a form user to digitally sign a form, the form developer must create a digital signature button, according to the language specs outlined in the UFDL button item description.

When the user signs the form, the user's digital signature is stored in a signature item (consisting of the signer identification, the encoded UFDL representation of what is being signed, and the filters applied). The developer does not create a signature item; rather, it is automatically created by the viewer or other form application when the user signs the form.

Once a form or a portion of a form has been signed, it cannot be altered. If it becomes altered, the signature will break, and all users will be notified of the broken signature. In addition, some applications may refuse to process a form containing a broken signature.

Example Signature Button and Signature Descriptions

```
empSigButton = new button
{
   type = "signature";
   value = signer;
   format = ["string", "mandatory"];
   signformat = "application/uwi_form;csp=\"Microsoft Base
Cryptographic Provider v1.0\";csptype=rsa_full;hashalg=sha1";
```

```
      signoptions = ["omit", "triggeritem", "coordinates"];
      signitemrefs = ["omit", "PAGE1.mgrSigButton",
    "PAGE1.admSigButton",
                              "PAGE1.empSignature",
      signature = "empSignature";
    }
    ...
     empSignature = new signature
      {
      signformat = "application/uwi_form;csp=\"Microsoft Base
    Cryptographic Provider v1.0\";csptype=rsa_full;hashalg=sha1";
               signer = "Jane D Smith, jsmith@insurance.com";
               signature = "PAGE1.empSignature";
               signitemrefs = ["omit", "PAGE1.mgrSigButton",
    "PAGE1.admSigButton",
                              "PAGE1.empSignature",
    "PAGE1.mgrSignature",
                              "PAGE1.admSignature"];
      signoptions = ["omit", "triggeritem", "coordinates"];
      mimedata = "MIIFMgYJKoZIhvcNAQcCoIIFIzCCBR8CAQExDzANBgkg"
      "AQUFADALB\ngkqhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl"
      "6hMrc5DySSP+X5j\nANfBGSOI\n9w0BAQQdWaYDVQQHEwhJbn"
      "Rlcm5ldDEXMBUGA1UEChM\nOVmVyaVNpZ24sIEluYy4xNDAKn"
      "1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmRdWFsIFN1YnNjcmliy"
      "ZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCARExETA";
    }
```

For more information, see the button and signature item descriptions.

2.7 Filters

UFDL supplies options for filtering transmissions and digital signatures. The filters allow the form developer to specify the items and options that should be included in or omitted from a transmission or a signed portion of the form.

For transmissions, filtering is a useful way to reduce file size. While compression reduces file size significantly, filters can be used to further optimize a transmission, by sending only the required data. Obviously, filtering needs to be done with care, as it is possible to destroy the layout or the original context of a form if it is applied without caution.

For information on compression, see the transmitformat option description.

For signatures, filtering is the method to use to allow portions of forms to be signed. For example, if you created a form that contained two sections, one for an employee to fill out and sign,

and one for an administrative officer to fill out and sign, you would use filters in each signature button to specify which portion of a form each signature applied to. For more details, see the button item description.

The filters for transmission are: transmitdatagroups, transmitgroups, transmititems, transmititemrefs, transmitoptions, and transmitoptionrefs.

The filters for digital signatures are: signdatagroups, signgroups, signitems, signitemrefs, signoptions, and signoptionrefs.

For example:

```
 submitButton = new button
 {
    type = "done";
    value = "Submit";
    url = ["http://www.server.dmn/cgi-bin/warehouse.exe"];
    transmititems = ["omit", "data"];
    transmitdatagroups = ["keep", "enclosures", "related"];
    transmititemrefs = ["omit", "page1.toolbar"];
    transmitoptions = ["omit", "bgcolor", "fontcolor"];
 }

 employeeSignatureButton = new button
 {
    type = "signature";
    value = "Sign This Section";
    signature = "empSignature";
    signitemrefs = ["keep", "page1.nameField", "page1.nameLabel",
                    "page1.dateField", "page1.dateLabel",
                    "page1.evaluationField",
                    "page1.evaluationLabel"];
 }
```

For details on each filter, see the option descriptions later in this document.

Order of Precedence of Filters

Within each family of filters, there are item filters and option filters. In addition, there are item type filters, which filter an entire type of items (all fields, for example), and item instance filters (which filter specific instances of items). The same filter levels exist for option filters.

The settings in filters are applied by a UFDL parser in the following manner:

Filter	Behavior		Notes
	If keep flag is used	If omit flag is used	
1.Filter types of items, based on transmititems / signitems setting	Keeps only those types referred to; throws others out, including their options	Omits only those types referred to; throws them out including their options	
2.Filter groups	Keeps those	Omits those	

of items based on transmitdatagroups and transmitgroups,	items whose tags are specified, even if the items are of a	items whose tags are specified, even if the items are of a type

or signdatagroups and signgroups settings	type that should not be kept according to a transmititems or signitems setting	that should be kept according to a transmititems or signitems setting	
3.Filter specific items based on transmititemrefs or signitemrefs settings	Keeps the items whose tags are specified; overrides previous setting if necessary	Omits the items whose tags are specified; overrides the previous settings if necessary	This option's settings override those in transmititems, transmitgroups, and transmitdatagroups or signitems, signgroups and signdatagroups
4.Filter types of options based on transmitoptions and signoptions setting	In the items that remain, keeps all option types referred to; throws others out	In the items that remain, omits all option types referred to	

5.Filter specific options based on transmitoptionrefs and signoptionrefs setting	Regardless of all other settings above, keeps the specific option instances referred to; does not keep any other options; in the case of items that will be omitted exept for a single option, the description will look like this:
	itemTag = new item
	{
	option = setting;
	}
		Regardless of all other settings above, omits the specific option instances referred to	This option's settings override all other filters (transmititems, transmitdatagroups, transmitgroups, transmititemrefs, transmitoptions or signitems, signdatagroups, signgroups, signitemrefs, signoptions)

Example

This example uses the transmit-family of options. The order of precedence would be the same for the sign-family of options.

```
     version = "4.0.0";

     page1 = new page
     {
```

```
    submitButton = new button
    {
            value = "Filter Submission";
            type = "done";
            url = ["http://www.server.dmn/cgi-bin/processForm"];
            transmititems = ["omit", "data"];
            transmitdatagroups = ["keep", "enclosures", "related"];
            transmititemrefs = ["omit", "page1.data2"];
            transmitoptions = ["omit" "filename"];
    }

    encloseButton = new button
    {
            image = "encloseImageData";
            type = "enclose";
            datagroup = ["enclosures", "related"];
    }

    data1 = new data
    {
            datagroup = ["enclosures"];
            filename = "jobdescr.frm";
            mimedata = "dfksdfsdfhsdhskdljhf";
    }

    data2 = new data
    {
            datagroup = ["related"];
            filename = "resume.doc";
            mimedata = "dfhsjdfsjhfjs";
    }

    encloseImageData = new data
    {
            filename = "c:\images\enclose.jpg";
            mimedata = "aswWWW8MjfbyhsUE&LKKELFir8dfd";
              "UUUmnskshie3mkjkkeiIIUIUOlfRlgdsoepgejgjj";
              "1sd\35fnnII\fjkess9Wfgjgkggkll\\rgakkk2klgjgkg";
    }

 }
```

As a result of the filtering, the following would happen (see result form description below):

- The encloseImageData data item would be stripped from the form, as a result of the transmititems setting.

- The data1 data item would remain in the form, as a result of the transmitdatagroups setting.

- The data2 data item would be stripped from the form, as a result

of the transmititemrefs setting.

- The filename option would be stripped from data1, as a result of the transmitoptions setting.

The form description that would be received once filtering was applied would look like this:

```
 version = "4.0.0";

 page1 = new page
 {
    submitButton = new button
    {
            value = "Filter Submission";
            type = "done";
            url = ["http://www.server.dmn/cgi-bin/processForm"];
            transmititems = ["omit", "data"];
            transmitdatagroups = ["keep", "enclosures", "related"];
            transmititemrefs = ["omit", "page1.data2"];
            transmitoptions = ["omit" "filename"];
    }

    encloseButton = new button
    {
            image = "encloseImageData";
            type = "enclose";
            datagroup = ["enclosures", "related"];
    }

    data1 = new data
    {
            datagroup = ["enclosures"];
            mimedata = "dfksdfsdfhsdhskdljhf";
    }

 }
------
```

2.8 Processing Forms

Once a user saves or submits a form, it becomes a form instance. In the course of a form instance's life cycle, it may be viewed by various users at various client sites. Also, several form processing applications may handle the form. To ensure consistency and integrity of the form's appearance and contents, there are some important form processing rules in UFDL.

UFDL offers two types of include statements:

- #include - Includes a file from the local drive in the form description, and, if the file does not exist, flags the error.

- #optinclude - Includes a file from the local drive if the file exists and, if the file does not exist, ignores it gracefully.

The rules governing handling of #include statements state:

- All #include and #optinclude statements are resolved when the form appears. The only exception occurs when the referenced file cannot be found. In this instance, the #include or #optinclude statement remains in the form.

- When a #include or #optinclude statement is resolved, the #include or #optinclude statement definition is permanently deleted from the form instance.

These rules combine to ensure that the definition of a particular form instance remains constant from first to last viewing, and that no user data disappears.

2.8b Expressions

The rule governing handling of expressions in value options states:

- Expressions are overwritten if the item is modifiable and the user updates the value displayed, or if the form submission format is HTML.

This rule ensures that forms submitted in UFDL format continue to work as originally designed even after processing.

---

3. UFDL Global and Page Settings

At the top of each form and each page, a form developer can specify options that apply to the whole form or the particular page. These are called global settings and page settings.

The syntax of global settings is as follows:

```
version = <version_number>;
<option definition2>
...
<option definitionn>
Notes:
i) The version option is mandatory. It must be the first line in
   the form.
ii)All options other than version are optional.
iii)Global settings must appear before the first page declaration.
iv)A page setting can override a form global option for the
   particular page.
```

The syntax of page settings is as follows:

```
<page tag> = new page
{
<option definition1>
...
<option definitionn>
Notes:
```

i) Page settings are optional.
ii)Page settings must appear before the first item definition in the page.
iii)A page will assume the characteristics specified in the global

settings unless they are overridden by settings of the same type in the page settings.

iv) Options within item declarations override page settings for the particular item they appear in.

The following pages outline which options can be used as global settings and which options can be used as page settings.

3.1 Global Settings

Global settings specify particular settings for the form and determine its physical characteristics. For example, the version option defines which version of UFDL the form was written in. The bgcolor option determines the background color of all pages in the form. Global settings appear at the top of a form and apply to the whole form. Options defined within a page or item can override global settings for that particular page or item.

Available Options

Option	Behavior
activated	Whether the form is displayed or not. Default: off
bgcolor	Background color of form. Default: 255, 255, 255 (white)
bordercolor	Border color of items in form. Default: 0, 0, 0 (black)
borderwid	Border width of items in form. Default: Depends on item
focused	Whether the form has the input focus (generally, if it's open, it does). Default: off
fontcolor	Color of all value text in form. Does not apply to the text of label options. Default: black
fontinfo	Style of all value text in form. Does not apply to the text of label options.
label	Text that appears in title bar of form. Default: n/a
next	Item the focus appears on when form opens. Default: First input item in form description
saveformat	Format in which the form is saved. Default: uncompressed UFDL

transmitformat	Format in which the form is transmitted. Default: uncompressed UFDL

triggeritem — Item tag of item that triggered a submit or done action. Default: n/a

version — Version of UFDL used to make the form. Default: n/a

Usage Notes

1) Define global settings at the top of the form, before the first page declaration.

2) The version option is mandatory and must be the first line in the form.

3) You can give the form a title that appears in the title bar by setting a global label option.

4) To specify a title to appear in the form's title bar, use the label option as a global setting.

Example

This example defines settings and characteristics for the form.

```
version = "3.2.0";
saveformat = "application/uwi_form; content-encoding=\"gzip\" ";
label = "Time Sheet";
bgcolor = ["ivory"];
fontinfo = ["Helvetica", "10", "plain"];
```

These global settings specify that:

- The form is written in version 3.2.0 of UFDL.
- All saves activated from the form should save the form as a compressed UFDL form, unless specified otherwise in an item that initiates a save.
- The title Time Sheet should appear in the title bar of all pages, unless specified otherwise in a page global.
- All pages, toolbars, boxes, labels, and tablets should have an ivory background, unless they contain an option specifying otherwise.
- All pages and items should use a plain, Helvetica, 10-point font, unless they contain an option specifying otherwise. (Note: Labels that are parts of other items, like fields, are excluded from the fontinfo option. They are set using the labelfontinfo option.)

3.2 Page Settings

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

items.

Available Options

Option	Behavior
activated	Whether the page is being displayed or not. Default: off
bgcolor	Background color of page. Default: 255, 255, 255 (white)
bordercolor	Border color of items in page. Default: 0, 0, 0 (black)
borderwidth	Border width of items in page. Default: Depends on item
fontcolor	Color of all value text in page. Does not apply to the text of label options. Default: black
focused	Whether the page has the input focus. (Generally, if it's open, it does). Default: off
fontinfo	Style of all value text in page. Does not apply to the text of label options. Default: Helvetica, 8, plain
label	Text that appears in title bar of page. Default: n/a
mouseover	Whether the mouse pointer is over the page.
next	Item the focus appears on when page opens. Default: First input item in form description
saveformat	Format in which the page is saved. Default: uncompressed UFDL
transmitformat	Format in which the page is transmitted. Default: uncompressed UFDL

Usage Notes

1) Define page settings at the top of a page, after the opening brace and before the first item declaration.
2) Page settings apply only to the page they are on.

3) Page settings are optional.

4) To specify a title to appear in the page's title bar, use the label option as a page setting.

Example

The following example shows page global options on two pages

within a single form.

...

```
 page_1 = new page
 {
    bgcolor = ["seashell"];
    next = "Name_field";

    <item declaration>
 ...
 }
 page_2 = new page
 {
    fontinfo = ["Helvetica", "14", "plain"];
    next = "Activity_popup";

    <item declaration>
 ...
```

Page one would have a seashell-colored background, and would direct the focus to the item called Name_field as soon as it opened. It would assume the rest of its settings from the form's global settings. (If no form global settings exist, the page will assume the UFDL defaults.)

On page two, the font would appear as plain, Helvetica, 14-point type and the focus would be directed to the item called Activity_popup as soon as the page opened. Page two would assume the rest of its settings from the page global options and UFDL defaults.

---

4. UFDL Form Items

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

```
 <item tag> = new <item type>
 {
  <option definition1>
  ...
  <option definitionn>
 }
 Notes:
 i) The braces are mandatory.
 ii) An item definition must begin on a new line.
 iii) Option definitions are optional.
 iv) You cannot assign values to options in other item definitions.
```

> Tip: UFDL is case sensitive. All item type names are lowercase.

The item tag uniquely identifies an item instance. No two item tags on a page can be the same. Item type is a name that identifies the

type of item. This section contains information about UFDL-defined item types and the options available for each.

Note: Defining an option more than once in an item's definition may cause unpredictable behavior.

See the section 'Item Definition' in section 2.4d for more information on the syntax and rules regarding an item definition.

4.1 action

The action item allows you to specify form-initiated actions that execute automatically. The actions can be any of the following types:

- link
- submit
- done
- display
- print
- cancel

See the type option section for a description of each of these actions.

You can define action items that occur only once or repeat at specified time intervals. You can also define actions that 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.

Available Options

Option	Behavior
activated	Specifies whether item is currently activated. Default: off
active	Specifies whether item is active or inactive. Default: on
data	Specifies a single data item associated with an action of type display. Default: n/a
datagroup	Identifies group or folder of enclosed files. Default: n/a
delay	Delays execution of automatic action or specifies automatic action repeat factor. Default: repeat factor of once, interval of zero seconds
transmitdatagroups	Lists which datagroups of items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmitformat	Specifies format of form data transmitted to

form processing application. Default:
uncompressed UFDL (application/uwi_form)

transmitgroups	Lists which groups of items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmititemrefs	Lists which specific items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmititems	Lists types of items to include in or omit from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
transmitoptionrefs	Lists which specific options should be kept or omitted from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
transmitoptions	Lists which types of options to include in or omit from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
type	Associates task type with item that can trigger a task: action, button, or cell. Default:link
url	Identifies an object to access, for items with type option setting of link, replace, submit, done, or pagedone. Default: n/a

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 a the page closes. Actions defined to occur before the page displays, occur each time the page opens.

Examples

Example 1

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

```
 sendStatus_action = new action
 {
  delay = ["repeat", "600"];
```

```
      type = "submit";
      url = ["http://www.server.com/cgi-bin/recv_status"];
     }
```

Example 2

This action will link to a search form as soon as the current page displays.

```
grabSearch_action = new action
{
 type = "link";
 url = ["http://www.server.com/application/ index/search.frm"];
}
```

Example 3

Background actions can also cancel forms, or prompt the user to save a form. Here is an example:

```
// Automatically prompt the user to save the form after 5 minutes
autoSave_action = new action
{
 delay = ["once", "300"];
 type = "save";
}
//
// Automatically close the form after 10 minutes
autoCancel_action = new action
{
 delay = ["once", "600"];
 type = "cancel";
}
```

4.2 box

Sample 1: Box

The box item creates a square box on the form. You may not place other items in the box; however, you may place other items on top of it. The purpose of box items is simply to add visual variety to the form.

Available Options

Option	Behavior
bgcolor	Defines background color of box. Default: page background color
bordercolor	Defines color of border around box. Default: black
borderwidth	Defines width of box's border in pixels. Default: zero pixels
fontinfo	Defines font name, point size, and font

characteristics for text portion of box.
Defaults: Helvetica, 8, plain

itemlocation	Specifies location of box in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
size	Specifies box's size in characters. Default: width 1 character, height 1 character

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).

Example

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

```
blue_box = new box
{
 bgcolor = ["blue"];
 size = ["25", "4"];
}
```

4.3 button

Sample 2: Button

The button item provides a click button to perform an action when selected. For example, you can use buttons to request data from a web server, submit or cancel the form, sign the form, save it to disk, or enclose external files.

Available Options

Option	Behavior
activated	Specifies whether the button is currently activated by user. Default: off
active	Specifies whether button is active or inactive. Default: on
bgcolor	Defines background color of button. Default: gray
bordercolor	Defines width of button's border in pixels. Default: zero pixels
borderwidth	Defines font name, point size, and font

characteristics for text portion of button.
Defaults: Helvetica, 8, plain

coordinates	Records position of mouse pointer on an image which must exist in a button. Default: n/a
data	Specifies a single data item associated with a button of type enclose, display, extract, or remove. Default: n/a
datagroup	Identifies group or folder of enclosed files. Default: n/a
focused	Specifies whether button has input focus. Default: off
fontcolor	Defines font color for text or filler portion of button. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of button. Default: Helvetica, 8, plain
format	Defines whether a button is mandatory or optional, for use with signature buttons. Default: optional (see also format option description)
help	Points to help message for button. Default: n/a
image	Associates an image with button. Default: n/a
itemlocation	Specifies location of button in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin.
justify	Aligns lines of text within space button occupies. Default: center
mouseover	Specifies whether mouse pointer is over button. Default: off
next	Identifies item to receive focus when user tabs ahead from current item. Default: depends on order in which page and item definitions occur within form definition
signature	Establishes UFDL item name by which a digital signature is identified. Default: n/a
signdatagroups	Specifies which datagroups are to be filtered for digital signature. Default: see "Order of

Precedence of Filters"

signer Adds user's common name and email address as

	they appear in user's personal certificate, identifying who signed the form. Default: n/a
signformat	Controls what system parameters are used to create the signature. Default: application/uwi_form
signgroups	Specifies which groups of items are to be filtered for digital signature. Default: see "Order of Precedence of Filters"
signitemrefs	Specifies which individual items are to be filtered for digital signature. Default: see "Order of Precedence of Filters"
signitems	Specifies which types of items are to be filtered for digital signature. Default: see "Order of Precedence of Filters"
signoptionrefs	Specifies which individual options are to be filtered for digital signature. Default: see "Order of Precedence of Filters"

signoptions	Specifies which types of options are to be filtered for digital signature. Default: see "Order of Precedence of Filters"
size	Specifies button's size in characters. Default: width of label, height of label
transmitdatagroups	Lists which datagroups of items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmitformat	Specifies format of form data transmitted to form processing application. Default: uncompressed UFDL
transmitgroups	Lists which groups of items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmititemrefs	Lists which specific items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmititems	Lists types of items to include in or omit from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
transmitoptionrefs	Lists which specific options should be kept or omitted from form data submitted to form processing application. Default: see "Order of Precedence of Filters"

transmitoptions	Lists which types of options to include in or omit from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
type	Associates task type with item that can trigger a task: action, button, or cell. Default:link
url	Identifies an object to access, for items with type option setting of link, replace, submit, done, or pagedone. Default: n/a
value	Contains text of button's label. Default: n/a

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 upate 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 so that you can force users to sign forms before submitting them. You do this by making a signature button mandatory, like this:

```
empSignatureButton = new button
{
  type = "signature";
  format = ["string", "mandatory"];
  value = signer;
  signer="";
  signature = "empSignature";
  signoptions = ["omit", "triggeritem", "coordinates"];
  signitemrefs = ["omit", "page1.empSignatureButton",
                  "page1.empSignature"];
}
```

There are two steps to making a signature button mandatory:

- Assign a format of ["string", "mandatory"].

- Set the button's value to equal the button's signer option setting.

By setting the format to mandatory, you specify that the button must have a value setting that is not empty before the user submits the form. By then equating the value to the setting of the signer

option, you make sure 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) You can set up a signature button to sign the whole form or just part of it. You do this by setting up filters on the signature, using the signdatagroups, signgroups, signitemrefs, signitems, signoptionrefs, and signoptions options. To learn about filtering, see "Filters"in section 2.7.

   Important:You should always at a minimum filter out the triggeritem and coordinates options. These options change when a submission is triggered or when a user clicks an image button, respectively. You should also consider filtering out any 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).

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 UFDL 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.

Examples

Example 1 - Link button

This button links the form to a server (www.server.com) and
 getHelp_button = new button

```
     {
      value = "Get Help";
      type = "link";
```

```
   url = ["http://www.server.com/application/help/formHelp.frm"];
  }
```

Example 2 - 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:
  submit_button = new button
  {
   value = "Process Form";
   fontinfo = ["Helvetica", "18", "bold", "italic"];
   type = "done";
   url = ["http://www.server.com/cgi-bin/formProcessor"];
  }
```

Example 3 - 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:
  enclose_button = new button
  {
   value = "Enclose File";
   fontinfo = ["Helvetica", "18", "bold", "italic"];
   type = "enclose";
   datagroup = ["Images_Asia", "Images_Eur" , "Images_SAmer"];
  }

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

Example 4 - Signature button

```
 This example shows a signature button with the signature item it
 creates when signed. The button contains the basic settings you
 should use. Its type is designated as signature; it omits
 triggeritem and coordinates options throughout to avoid breaking
 the signature.
  empSigButton = new button
  {
     type = "signature";
     value = signer;
     format = ["string", "mandatory"];
     signformat = "application/uwi_form;csp=\"Microsoft Base
  Cryptographic Provider v1.0\";csptype=rsa_full;hashalg=sha1";
     signoptions = ["omit", "triggeritem", "coordinates"];
     signitemrefs = ["omit", "PAGE1.mgrSigButton",
  "PAGE1.admSigButton", "PAGE1.empSignature",
  "PAGE1.mgrSignature", "PAGE1.admSignature"];
```

```
        signature = "empSignature";
    }
    ...
     empSignature = new signature
    {
```

```
        signformat = "application/uwi_form;csp=\"Microsoft Base
     Cryptographic Provider v1.0\";csptype=rsa_full;hashalg=sha1";
                signer = "Jane D Smith, jsmith@insurance.com";
                signature = "PAGE1.empSignature";
                signitemrefs = ["omit", "PAGE1.mgrSigButton",
     "PAGE1.admSigButton", "PAGE1.empSignature",
     "PAGE1.mgrSignature", "PAGE1.admSignature"];
                signoptions = ["omit", "triggeritem", "coordinates"];
              mimedata = "MIIFMgYJKoZIhvcNAQcCoIIFIzCCBR8CAQExDzANBgkg"
                "AQUFADALB\ngkqhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl"
                "6hMrc5DySSP+X5j\nANfBGSOI\n9w0BAQQwDwYDVQQHEwhJbn"
                "Rlcm5ldDEXMBUGA1UEChM\nOVmVyaVNpZ24sIEluYy4xNDAKn"
                "1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmRdWFsIFN1YnNjcmliy"
                "ZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCARExETA";
     }
------
```

4.4 cell

The cell item 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.

Available Options

Option	Behavior
activated	Specifies whether cell is currently activated by user. Default: off
active	Specifies whether cell is active or inactive. Default: on
data	Specifies a single data item to associate with a cell of type enclose, display, extract, or remove. Default: n/a
datagroup	Identifies group or folder of enclosed files. Default: n/a

group	Groups cells together. Default: n/a

label	Specifies external text label for cell. Default: n/a
transmitdatagroups	Lists which datagroups of items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmitformat	Specifies format of form data submitted to processing application. Default: uncompressed UFDL
transmitgroups	Lists which groups of items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmititemrefs	Lists which specific items should be kept or omitted from a transmission. Default: see "Order of Precedence of Filters"
transmititems	Lists types of items to include in or omit from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
transmitoptionrefs	Lists which specific options should be kept or omitted from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
transmitoptions	Lists which options to include in or omit from form data submitted to form processing application. Default: see "Order of Precedence of Filters"
type	Associates task type with item that can trigger a task: action, button, or cell. Default:link
url	Identifies an object to access, for items with type option setting of link, replace, submit, done, or pagedone. Default: n/a
value	Contains identity of most recently selected cell

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.

```
countryPopup = new popup
{
   label = "Country";
```

```
        group = "country";
        format = ["string", "mandatory"];
      }
      albCell = new cell
```

```
    {
       value = "Albania";
       type = "select";
    }
    algCell = new cell
    {
       value = "Algeria";
       group = "country";
       type = "select";
    }
    banCell = new cell
    {
       value = "Bangladesh";
       group = "country";
       type = "select";
    }
```

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, you need to dereference it, like this:
<page_tag>.<list_tag>.value->value

For example:
page1.countryPopup.value->value

When a user selects a cell from a list, the item tag of the cell is stored as the value of the list. Hence the dereference syntax.

4.5 check

Sample 3: Check Box

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

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

box for the users of each platform.

Available Options

Option	Behavior
active	Specifies whether check box is active or inactive. Default: on
bgcolor	Defines background color of check box. Default: white
bordercolor	Defines color of border around check box. Default: black
editstate	Defines one of three possible edit states for modifiable items: readonly, writeonly, or readwrite. Default: readwrite.
focused	Specifies whether check box currently has input focus. Default: off
fontcolor	Defines font color for text or filler portion of button. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of button. Defaults: Helvetica, 8, plain
help	Points to help message for button. Default: n/a
itemlocation	Specifies location of button in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
label	Specifies external text label for check box. Default: n/a
labelbgcolor	Defines background color for label specified in label option. Default: in toolbar, background color of toolbar; otherwise, background color of page
labelbordercolor	Defines color of border around label specified in label option. Default: black
labelborderwidth	Defines width of border around label specified in label option, in pixels. Default: zero pixels
labelfontcolor	Defines font color for label specified in label option. Default: black

labelfontinfo	Defines font name, point size, and font characteristics for label specified in label option. Default: Helvetica, 8, plain

mouseover	Specifies whether mouse pointer is over item. Default: off
next	Identifies item to receive focus when user tabs ahead from current item. Default: depends on order in which page and item definitions occur within form definition
size	Specifies check box's size in characters. Default: width 1 character, height 1 character
value	Indicates user answer's as to whether check box is checked (on) or not (off). Default: off

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 boxes 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).

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.

```
healthPlan_check = new check
{
 value = "on";
 label = "Activate Health Plan";
 labelfontinfo = ["Times", "14", "bold"];
 labelfontcolor = ["blue"];
}
```

4.6 combobox

Sample 4: Combobox

Comboboxes act like a hybrid of a field and a popup. Unopened, a

combobox with a label occupies the same space 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.

Available Options

Option	Behavior
activated	Specifies whether the pop-down list is currently activated. Default: off
active	Specifies whether combobox is active or inactive. Default: on
bgcolor	Defines background color of combobox. Default: white
bordercolor	Defines color of border around combobox. Default: black
editstate	Defines one of three possible edit states for modifiable items: readonly, writeonly, or readwrite. Default: readwrite.
focused	Specifies whether the combobox currently has the input focus. Default: off
fontcolor	Defines font color for text or filler portion of combobox. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of combobox. Defaults: Helvetica, 8, plain
format	Applies to value of each cell linked to combobox, flagging or filtering cells that fail check, and replacing value of cells that pass with formatted value. Default: format option
group	Groups comboboxes together. Default: n/a
help	Points to help message for combobox. Default: n/a
itemlocation	Specifies location of combobox in page layout. Default: in body of page, under previous item

in page definition, aligned along page's
left margin

label	Specifies external text label for combobox. Default: n/a
labelbgcolor	Defines background color for label specified in label option. Default: in toolbar, background color of toolbar; otherwise, background color of page
labelbordercolor	Defines color of border around label specified in label option. Default: black
labelborderwidth	Defines width of border around label specified in label option, in pixels. Default: zero pixels
labelfontcolor	Defines font color for label specified in label option. Default: black
labelfontinfo	Defines font name, point size, and font characteristics for label specified in label option. Default: Helvetica, 8, plain
mouseover	Specifies whether the mouse pointer is over the combobox. Default: off
next	Identifies item to receive focus when user tabs ahead from current item. Default: depends on order in which page and item definitions occur within form definition
previous	Identifies the item to receive the focus when the user tabs backwards from the current item. Default: previous item within form definition
size	Specifies combobox's size in characters. Default: width=larger of label width and widest cell, height=1 character
value	Contains one of following: value of most recently chosen selection, nothing if an action was most recently chosen, or text entered if something was typed in most recently

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,

you need to dereference it, like this:
<page_tag>.<list_tag>.value->value
For example:
page1.countryPopup.value->value

When a user selects a cell from a list, the item tag of the cell is stored as the value of the list. Hence the dereference syntax.

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.

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

```
CATEGORY_POPUP = new combobox
{
 group = "combo_Group";
```

```
 label = "Choose a Color:";
}
```

Notice 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. Notice they are select cells and they belong to the same group as the combobox: combo_Group.

```
 RED_CELL = new cell
 {
  group = "combo_Group";
  type = "select";
  value = "Red";
 }
 WHITE_CELL = new cell
 {
  group = "combo_Group";
  type = "select";
  value = "White";
 }
 BLUE_CELL = new cell
 {
  group = "combo_Group";
  type = "select";
  value = "Blue";
 }
```

4.7 data

The data item stores an information object such as an image, a sound, or an enclosed file in a UFDL form. Data in data items must be encoded in base64 format.

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

Available Options

Option	Behavior
datagroup	Identifies group or folder of enclosed files. Default: n/a
filename	Identifies name of enclosed file. Default: n/a
mimedata	Contains actual data associated with a data item. Default: n/a
mimetype	Defines MIME type of data stored in a data item

Usage Notes

1) See the section 'Binary Data' on page 41 for more information on binary data in UFDL forms.

2) Store the data in the mimedata option, and the data's MIME type in the mimetype option.

3) If a button or cell of type enclose contains a data option that points to a data item (as opposed to using the datagroup option), then special rules apply to the data item's behavior. If a user encloses a new data item using that button, the new information overwrites the old. For example, if the data item originally contained a jpeg image of a dog, and then a user enclosed a png image of a house, then the data item's mimedata, mimetype, and filename options update themselves to contain the information about the house image.

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). Notice the quotation marks surrounding each segment of the data.

```
 Supporting_Documents_1 = new data
 {
  filename = "smithltr.doc";
  mimetype = "application/uwi_bin";
  mimedata =
 "R0lGODdhYABPAPAAAP///wAAACwAAAAAYABPAAAC/4SPqcvtD02Y"
 "Art68+Y7im7ku2KkzXnOzh9v7qNw+k+TbDoLFTvCSPzMrS2YzmTE+p"
 "yai3YUk9R6hee2JFP2stju+uG0ptvdeKptb+cX8wfY1jdYU4ehKDi3pdJw"
 "44yAJEqcW28cA5M0oEKnqKasZwydrK9Wo6JTtLG9p5iwtWi8Tbi/b7E0"
 "rvKixzbHJyrDq2uNggaXUs1NlLi36AW3AGv7VWhIPA7TzvdOGi/vvr0Of"
 "ft3Nrx89JewCQJYTirxi2PwgnRpNoMV5FIIboOnqTszFLFIMhQVI0yOz";
 }
```

4.8 field

Sample 5: Field

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

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

Available Options

Option	Behavior
active	Specifies whether field is active or inactive. Default: on

bgcolor	Defines background color of field.

	Default: white
bordercolor	Defines color of border around field. Default: black
editstate	Defines one of three possible edit states for modifiable items: readonly, writeonly, or readwrite. Default: readwrite.
focused	Specifies whether the field has the input focus. Default: off
fontcolor	Defines font color for text or filler portion of field. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of field. Defaults: Helvetica, 8, plain
format	Specifies data type of field's data, along with flags allowing you to specify edit checks and formatting you want applied to data. Default: n/a
help	Points to help message for field. Default: n/a
itemlocation	Specifies location of field in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
justify	Aligns lines of text within the space field occupies. Default: left
label	Specifies external text label for field. Default: n/a
labelbgcolor	Defines background color for label specified in label option. Default: in toolbar, background color of toolbar; otherwise, background color of page
labelbordercolor	Defines color of border around label specified in label option. Default: black
labelborderwidth	Defines width of border around label specified in label option, in pixels. Default: zero pixels
labelfontcolor	Defines font color for label specified in label option. Default: black

labelfontinfo	Defines font name, point size, and font characteristics for label specified in label option. Default: Helvetica, 8, plain

mouseover	Specifies whether the mouse pointer is over the field. Default: off
next	Identifies item to receive focus when user tabs ahead from current item. Default: depends on order in which page and item definitions occur within form definition
size	Specifies field's size in characters. Default: width 30 characters, height 1 character
value	Reflects contents of field-numeric, alphabetic, or otherwise. Default: n/a

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 allowing you to specify edit checks and formatting you want applied 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.

Examples

Example 1

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.

```
 income_field = new field
 {
  label = "Annual income";
  value = "23000";
  size = ["20", "1"];
  fontinfo = ["Courier", "12", "plain"];
  labelfontinfo = ["Helvetica", "12", "plain"];
```

```
      labelfontcolor = ["blue"];
     }
```

Example 2

To create a multiple line field, the vertical size of the field must be adjusted (either with size or with itemlocation modifiers). As well, vertical scroll bars can be added, and word wrapping turned on. Here is an example:

```
job_field = new field
{
 label = "Job Description";
 size = ["50", "5"];
 scrollvert = "always";
 scrollhoriz = "wordwrap";
 fontinfo = ["Times", "12", "plain]";
 labelfontinfo = ["Helvetica", "12", "plain"];
 labelfontcolor = ["blue"];
}
```

4.9 help

A help item defines a help message you can use to support various external items in the form. You can create a separate help item for every item you want to support, or you can use one help item for several items.

Available Options

Option	Behavior
active	Specifies whether field is active or inactive. Default: on
value	Reflects help item's contents

Usage Notes

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

2) The link between the help item and the supported item is created by the help option in the supported item's definition. The help option contains the help item's item reference.

Example

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

First, here is the button definition. Notice the help item's item reference in the help option.

```
fullPicture_button = new button
{
```

```
      help = "button_help";
      fontinfo = ["Times", "14", "plain"];
```

```
      type = "link";
      url = ["http://www.server.com/application/fullPic.frm"];
     }

    Now, here is 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.

     button_help = new help
     {
      value = "Pressing this button will bring a full-sized image in
               a form "
                "down to your viewer.";
     }
------
```

4.10 label

Sample 6: Label

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

Available Options

Option	Behavior
active	Specifies whether label is active or inactive. Default: on
bgcolor	Defines background color of label. Default: transparent
bordercolor	Defines color of border around label. Default: black
fontcolor	Defines font color for text or filler portion of label. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of label. Defaults: Helvetica, 8, plain
format	Specifies data type of label's data, along with flags allowing you to specify edit checks and formatting you want applied to data
help	Points to help message for label.
image	Defines image for label. Default: n/a

itemlocation	Specifies location of label in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin

justify	Aligns lines of text within space label occupies
size	Specifies label's size in characters. Default: width 1 character if label empty, otherwise label width; height 1 character if label empty, otherwise label height
value	Defines text for label. Default: n/a

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 upate 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, if you wanted to place a label inside a colored box, in order to make a title section that stands out, you could do so without specifying a background color for the label:

```
box = new box
{
   size = ["30", "3"];
}
label = new label
{
   itemlocation = [["alignhorizc2c", "box"],
                           ["alignvertc2c", "box"]];
   value = "Great Insurance";
   fontcolor = ["white"];
   fontinfo = ["Helvetica", "14", "bold"];
// Note: you could also specify bgcolor = ["transparent"], but you
         don't
// need to because the default is transparent.
}
```

Examples

Example 1
This is an example of a text label. The text is centered in the

space the label occupies. The label width is 30 characters (thus it is bigger than the text in the label).

```
 MAINMENU_LABEL = new label
 {
  value = "Welcome to the Main Menu";
  fontinfo = ["Helvetica", "24", "bold", "italic"];
  size = ["30", "1"];
  justify = "center";
 }
```

Example 2

This is an example of a multiple line text label. Notice the line break escape sequences indicating the end of each line.

```
 // Specify right justification for this label.
 RHYME_LABEL = new label
 {
  value = "Little miss Muffet\n Sat on her tuffet,\n"
            "Eating her curds and whey.\n When along came a
             spider,\n"
            "who sat down beside her,\n and frightened miss
             Muffet away!";
  fontinfo = ["Times", "16", "italic"];
 }
```

4.11 line

Sample 7: Line

The line item draws a simple vertical or horizontal line on the form. This is useful when you want to visually separate parts of a page.

Available Options

Option	Behavior
fontcolor	Defines font color for text or filler portion of label. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of label. Defaults: Helvetica, 8, plain
itemlocation	Specifies location of line in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
size	Determines whether line is horizontal or vertical: if horizonal dimension=0 then line is vertical, if vetical dimension=0 then line is

	horizontal; calculated in characters
thickness	Determines how thick line will be, in pixels

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.
   - The thickness option determines how thick the line will be. Thickness is calculated in pixels.

2) 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.
3) The fontcolor option defines the color of the line.

Example

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

```
BLUE_LINE = new line
{
 size = ["40", "0"];
 thickness = "5";
}
```

4.12 list

Sample 8: List

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

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

Available Options

Option	Behavior
active	Specifies whether list is active or inactive. Default: on
bgcolor	Defines background color of list. Default: white
bordercolor	Defines color of border around list. Default: black
editstate	Defines one of three possible edit states

for modifiable items: readonly, writeonly,
or readwrite. Default: readwrite.

focused	Specifies whether the list has the input focus. Default: off
fontcolor	Defines font color for text or filler portion of list. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of list. Defaults: Helvetica, 8, plain
format	Applies to value of each cell linked to list, flagging or filtering cells that fail check, and replacing value of cells that pass with formatted value. See format option
help	Points to help message for list.
itemlocation	Specifies location of list in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
label	Specifies external text label for list. Default: n/a
labelbgcolor	Defines background color for label specified in label option. Default: in toolbar, background color of toolbar; otherwise, background color of page
labelbordercolor	Defines color of border around label specified in label option. Default: black
labelborderwidth	Defines width of border around label specified in label option, in pixels. Default: zero pixels
labelfontcolor	Defines font color for label specified in label option. Default: black
labelfontinfo	Defines font name, point size, and font characteristics for label specified in label option. Default: Helvetica, 8, plain
mouseover	Specifies whether the mouse pointer is over the list. Default: off
next	Identifies item to receive focus when user tabs ahead from current item. Default: depends on order in which page and item definitions occur within form definition

size	Specifies list's size in characters. Default: width=larger of label width and widest cell, height=number of cells in list
value	Contains item reference of most recently selected cell in list (if it was a select cell); contains nothing if most recently selected cell was not a select cell

Usage Notes

1) Place cells in a list by creating a group for the list and assigning cells to the group. Create a group using the group option in the list definition. Assign cells to the group using the group option in the cell definition.

2) 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, you need to dereference it, like this:
<page_tag>.<list_tag>.value->value
For example:
page1.countryPopup.value->value

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.

Example

This is an example of a list containing three actions: submit form,

save form, and cancel form.

Here is the list definition.

```
MAINMENU_LIST = new list
{
 group = "list_Group";
 label = "Options Menu";
 labelfontcolor = ["blue"];
 size = ["3", "20"];
}
```

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

```
 SUBMIT_CELL = new cell
 {
  group = "list_Group";
  type = "submit";
  url = ["http://www.server.com/cgi-bin/processForm"];
  value = "Submit Form";
 }
 SAVE_CELL = new cell
 {
  group = "list_Group";
  type = "save";
  value = "Save Form";
 }
 CANCEL_CELL = new cell
 {
  group = "list_Group";
  type = "cancel";
  value = "Cancel this Form";
 }
```

4.13 popup

Sample 9: Popup Menu

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

The entries in the popup are cell items. Selections are cells with a type option setting of select. Actions are cells with any other type option setting.
Popups act like a hybrid of a label, a button, and a list.
Unopened, a popup occupies only the space required for its label.
Open, the popup displays a list of selections and actions. After a user chooses a selection or an action, the popup closes (that is,

returns to its unopened state). A popup's label displays inside the popup item.

Available Options

Option	Behavior
activated	Specifies whether the popup list is "popped up". Default: off
active	Specifies whether popup is active or inactive. Default: on
bgcolor	Defines background color of popup. Default: white
bordercolor	Defines color of border around popup. Default: black
borderwidth	Defines width of popup's border, in pixels. Default: one pixel
editstate	Defines one of three possible edit states for modifiable items: readonly, writeonly, or readwrite. Default: readwrite.
focused	Specifies whether the popup has the input focus. Default: off
fontcolor	Defines font color for text or filler portion of popup. Default: black
fontinfo	Defines font name, point size, and font characteristics for text portion of popup. Defaults: Helvetica, 8, plain
group	Groups cells in popup together.
help	Points to help message for popup.
itemlocation	Specifies location of popup in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
justify	Aligns lines of text within the space popup occupies.
label	Specifies external text label for popup. Default: n/a
mouseover	Specifies whether the mouse pointer is over the popup. Default: off
next	Identifies item to receive focus when user tabs ahead from current item. Default: depends on order in which page and item definitions occur within form definition

size	Specifies popup's size in characters. Default: width=larger of label width and widest

	cell, height=1 character
value	Contains item reference of most recently selected cell in popup (if it was a select cell); contains nothing if most recently selected cell was not a select cell

Usage Notes

1) Place cells in a popup by creating a group for the popup and assigning cells to the group. Create a group using the group option in the popup definition. Assign cells to the group using the group option in the cell definition.

2) 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, you need to dereference it, like this:
<page_tag>.<list_tag>.value->value

For example:
page1.countryPopup.value->value

When a user selects a cell from a list, the item tag of the cell is stored as the value of the list. Hence the dereference syntax.

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.

Example

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

Here is the popup definition. Notice 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.

```
CATEGORY_POPUP = new popup
{
 group = "popup_Group";
 label = "Choose a Category:";
}
```

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

```
HISTORY_CELL = new cell
{
 group = "popup_Group";
 type = "select";
 value = "World History";
}
SCIENCE_CELL = new cell
{
 group = "popup_Group";
 type = "select";
 value = "Physical Sciences";
}
MUSIC_CELL = new cell
{
 group = "popup_Group";
 type = "select";
 value = "Music";
}
```

4.14 radio

Sample 10: Radio Buttons

The radio button item is 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.

Available Options

Option	Behavior
active	Specifies whether radio button is active or inactive. Default: on
bgcolor	Defines background color of radio button. Default: white
bordercolor	Defines color of border around radio button. Default: black
borderwidth	Defines width of radio button's border, in pixels. Default: 1 pixel
editstate	Defines one of three possible edit states for modifiable items: readonly, writeonly, or readwrite. Default: readwrite.
focused	Specifies whether the radio button has the input focus. Default: off
fontcolor	Determines color of radio button fill pattern. Default: red
fontinfo	Defines font name, point size, and font characteristics for text portion of radio button. Defaults: Helvetica, 8, plain
group	Groups radio buttons together.
help	Points to help message for radio button.
itemlocation	Specifies location of radio button in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
label	Defines label to appear above radio button and aligned with left edge.
mouseover	Specifies whether the mouse pointer is over the radio button. Default: off
next	Identifies item to receive focus when user tabs ahead from current item. Default: depends on order in which page and item definitions occur within form definition
size	Specifies radio button's size in characters. Default: width 1 character, height 1 character
value	Indicates radio button's status: on indicates

chosen, off indicates not chosen.
Default: not chosen

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).

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.

```
NAME_RADIO = new radio
{
 value = "on";
 group = "search_Group";
 label = "Search by Name";
}
NUMBER_RADIO = new radio
{
 group = "search_Group";
 label = "Search by Number";
}
OCCUPATION_RADIO = new radio
{
 group = "search_Group";
 label = "Search by Occupation";
}
```

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.

4.15 signature

The signature item 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.

Available Options

Option	Behavior
mimedata	Contains actual data associated with signature. Default: n/a
signature	Identifies the button that created the signature. Default: n/a
signdatagroups	Identifies group or folder of enclosed files to be filtered for signature. Default: "Order of Precedence of Filters"
signer	Adds text similar to user's email signature, identifying who signed form. Default: n/a
signformat	Controls what system parameters are used to create the signature. Default: application/uwi_form
signgroups	Identifies groups of items to be filtered for signature. Default: "Order of Precedence of Filters"
signitemrefs	Identifies item references to be filtered for signature. Default: "Order of Precedence of Filters"
signitems	Identifies type of items to be filtered for signature. Default: "Order of Precedence of Filters"
signoptionrefs	Identifies option references to be filtered for signature. Default: "Order of Precedence of Filters"
signoptions	Identifies type of options to be filtered for signature. Default: "Order of Precedence of Filters"

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 UFDL 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 allow you 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. For details on the order in which filters are applied, see "Order of Precedence of Filters"

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

Example

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

```
empSigButton = new button
{
   type = "signature";
   value = signer;
   format = ["string", "mandatory"];
   signformat = "application/uwi_form;csp=\"Microsoft Base
Cryptographic Provider v1.0\";csptype=rsa_full;hashalg=sha1";
signoptions = ["omit", "triggeritem", "coordinates"];
   signitemrefs = ["omit", "PAGE1.mgrSigButton",
                "PAGE1.admSigButton","PAGE1.empSignature",
                "PAGE1.mgrSignature","PAGE1.admSignature"];
   signature = "empSignature";
}
```

```
     ...
     empSignature = new signature
      {
                signformat = "application/uwi_form;csp=\"Microsoft Base
```

```
     Cryptographic Provider v1.0\";csptype=rsa_full;hashalg=sha1";
                signer = "Jane D Smith, jsmith@insurance.com";
                signature = "PAGE1.empSignature";
                signitemrefs = ["omit", "PAGE1.mgrSigButton",
                        "PAGE1.admSigButton","PAGE1.empSignature",
                        "PAGE1.mgrSignature", "PAGE1.admSignature"];
                signoptions = ["omit", "triggeritem", "coordinates"];
               mimedata ="MIIFMgYJKoZIhvcNAQcCoIIFIzCCBR8CAQExDzANBgkg"
                 "AQUFADALB\ngkqhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl"
                 "6hMrc5DySSP+X5j\nANfBGSOI\n9w0BAQQwDwYDVQQHEwhJbn"
                 "Rlcm5ldDEXMBUGA1UEChM\nOVmVyaVNpZ24sIEluYy4xNDAKn"
                 "1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmRdWFsIFN1YnNjcmliy"
                 "ZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCARExETA";
------
```

4.16 spacer

The spacer item creates space between items on a form. It can be any size you specify. It is invisible.

Available Options

Option	Behavior
fontinfo	Defines font name, point size, and font characteristics for label of spacer. Defaults: Helvetica, 8, plain
itemlocation	Specifies location of radio button in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
label	Determines size of spacer, though not visible.
size	Specifies spacer's size in characters. Default: width=1 character if label empty, otherwise label width, height=1 character if label empty, otherwise label height

Usage Notes

1) You can size a spacer 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 you use a label, the spacer equals the size of the text you type 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).

Example

Example 1

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

```
 3_SPACER = new spacer
 {
  size = ["1", "3"];
 }
```

Example 2

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

```
 WELCOME_SPACER = new spacer
 {
  label = "Welcome to Information Line";
 }
```

4.17 tablet

Sample 11: Tablet

The tablet item creates a rectangular space or drawing object on the page where users can draw or write using the mouse pointer. This allows users to do such things as sign the form.

To draw on a tablet, users hold down the left mouse button while moving the mouse pointer over the space. To erase the marks, users position the mouse over the tablet, hold down CONTROL and click the right mouse button.

The tablet's background may be blank or composed of an image. The user draws on the background.

Available Options

Option	Behavior
active	Specifies whether tablet is active or inactive. Default: on
bgcolor	Defines background color of radio button. Default: page background color
bordercolor	Defines color of border around tablet. Default: black
borderwidth	Defines width of tablet's border, in pixels. Default: 1 pixel
editstate	Defines one of three possible edit states for modifiable items: readonly, writeonly, or

readwrite. Default: readwrite.

fontcolor	Determines pen color

fontinfo	Defines font name, point size, and font characteristics for text portion of tablet. Defaults: Helvetica, 8, plain
help	Points to help message for tablet.
image	Associates tablet with data item. Default: n/a
itemlocation	Specifies location of tablet in page layout. Default: in body of page, under previous item in page definition, aligned along page's left margin
justify	Aligns lines of text within space tablet occupies.
mouseover	Specifies whether the mouse pointer is over the tablet. Default: off
size	Specifies tablet's size in characters. Default: width 1 character, height 1 character
value	Used to set initial size of tablet; otherwise, tablet sizes itself to size of text entered in value

Usage Notes

1) A tablet item must contain an image option that associates it with a data item. The data item must also exist in the form. The user's drawing marks will be stored as image data in the data item.

2) For example, this is the code necessary to create a blank tablet (that contains no image background) on a form.

```
sketch_tablet = new tablet
{
 fontcolor = ["blue"];
 size = ["30", "5"];
 image = "sketch_data";
}
...
sketch_data = new data
{
}
```

3) To place an image in a tablet's background, store the image data in the data item already associated with the tablet. Note that when the user draws on the tablet, the user's marks will be stored as part of the same image.

For example, this piece of sample form code shows a tablet that

```
    contains an image called sign_data.
     sign_tablet = new tablet
     {
```

```
        fontcolor = ["blue"];
        size = ["30", "5"];
        image = "sign_logo";
     }
     ...
     sign_data = new data
     {
        mimedata = "R0lGODdhYABPAPAAAP///wAAACwAAAAAYABA/"
     "Art68+Y7im7ku2KkzXnOzh9v7qNw+k+TbDoLFTvCSPzMrSzTE+p"
     "yai3YUk9R6hee2JFP2stju+uG0ptvdeKptb+cX8wfY1jdYU4KpdJw"
     "44yAJEqcW28cA5M0oEKnqKasZwydrK9Wo6JTt9p5iwt8bi/b7E0"
     "rvKixzbHJyrDq2uNggaXUs1NlLi36AW3AGv7VWhIPAzvdGi/vvr0Of"
     "ft3Nrx89JewCQJYTirxi2PwgnRpNoMV5FIIboOnqTszFMhVI0yOz";
     }
```

4) The fontcolor option determines the pen color.

5) The pen width is two pixels.

6) The value can be used to set the initial size of the tablet. If no size is indicated, and no mimedata exists, the tablet will size itself to the size of the text entered in the value.

7) If an enclosure mechanism is used to replace an image stored in a data item with a new image, then buttons, labels, and tablets 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.

Example

This example shows a blank tablet with a background color of pale green. It is 40 characters wide and 10 characters high.

```
 users_signature = new tablet
 {
  bgcolor = ["PaleGreen"];
  size = ["40", "10"];
  fontinfo = ["Courier", "12"];    // This governs the size.
  fontcolor = ["black"];           // This governs pen color.
  image = "signature_data";
 }
 signature_data = new data
 {
 }
------
```

4.18 toolbar

Sample 12: Toolbar

The toolbar item allows you to define a toolbar for your 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, you place items in the toolbar that you want users 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, you will have to scroll to see everything it contains.

Refer to the section 'Toolbars' for more information on toolbars.

Available Options

You can use the following option with toolbar:

Option	Behavior
bgcolor	The background color of the toolbar. Default: background color of page.
mouseover	Specifies whether the mouse pointer is over the toolbar. Default: off

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.

Example

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

Here is the toolbar definition:

```
TOOL_BAR = new toolbar
{
 bgcolor = ["cornsilk"];
}
```

Here are the items belonging to the toolbar.

```
COMPANY_NAME = new label
{
 value = "My Company";
 itemlocation = [["within", "TOOL_BAR"]];
}
TB_SPACER = new spacer
{
 itemlocation = [["within", "TOOL_BAR"], ["below", "COMPANY_NAME"]];
}
SUBMIT_BUTTON  = new button
{
 value = "Submit Form";
 type = "submit";
```

```
 url = ["http://www.server.com/cgi-bin/formProcessor"];
 itemlocation = [["within", "TOOL_BAR"], ["below", "TB_SPACER"]];
}
```

```
     CANCEL_BUTTON  = new button
     {
      type = "cancel";
      itemlocation = [["within","TOOL_BAR"], ["after", "SUBMIT_BUTTON"]];
     }
------
```

4.19 <custom item>

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

Available Options

You can use all UFDL options and any custom options with custom items.

Usage Notes

1) The naming conventions for a custom item are as follows:
   - It must begin with an alphabetic character.
   - It can contain any of the characters A-Z, a-z, 0-9, $ and underscore.
   - It must contain an underscore.

Example

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

```
     STATUS_EVENT = new ma_event
     {
      active = "off";
      ma_id = "UF45567 \t /home/users/preferences01";
     }
------
```

5. UFDL Form Options

An option defines a characteristic of a form, a page, or an item. An option definition is an assignment statement. The expression on the right hand side of the equal sign contains the option's setting. The syntax of an option definition statement is as follows:

```
     <option identifier> = <expression>;
     Note: The semicolon is mandatory and terminates the statement.
```

Option identifier is a name that identifies the type of option. It can be a UFDL-defined option or a custom option. Examples of option

identifier are: bgcolor, fontinfo, itemlocation, and size. See the following pages for a description of each option and its possible values.

An expression specifies a value. An expression can be any of the following:

- a literal
- a reference to another option definition in the form
- an operation
- an array specification

Use an array specification for options requiring or permitting multiple values. The syntax of an array specification is as follows: [PD1]

```
[<element1>, <element2>, ... <elementn>]
Note: 'n' is the number of settings in the option.
```

An element can be any of the following:

- an expression
- an element definition statement

The brackets surrounding the array specification are mandatory even when there is only one element in the list.

The evaluation of array elements is done in their order of position unless the elements have UFDL-defined variable names. See the following section for a discussion of variable names.

Element Definition Statements

The element definition statement allows you to assign a variable name to an array element. Variable names permit you to refer to the element by name rather than by its position in the array. The syntax of an element definition statement is:

```
<variable> = <expression>
```

See the section 'Option Definition' for more information on expressions and arrays.

Characteristics

Options set for the form or a page are called characteristics. Form characteristics are global to the entire form. Page characteristics are global to the page on which they occur.

Defining Form Characteristics

Defining form characteristics is optional. It has the effect of setting characteristics that are global to the form. These characteristics override the defaults defined by UFDL. Specific pages or items will override these global characteristics if the same option is set differently for that page or item.

Use the reference global.global when referring to form

characteristics.

Defining Page Characteristics

Defining page characteristics is also optional. It has the effect of setting characteristics that are global to the page. These settings override the defaults defined by UFDL and any form characteristics. Specific pages or items will override these global characteristics if the same option is set differently for that page or item..

Use the reference global or <page tag>.global when referring to page characteristics.

Item Reference

An item reference identifies a particular item instance. The syntax of an item reference is as follows:

<item tag>
- for items on another page
<page tag>.<item tag>

Data Type Designators

UFDL defines a set of data types to describe the variable data in option settings. Each option's description includes the necessary data type information.

UFDL uses the following data type designators:

Data Type	Description
char	a single ASCII character
string	a series of ASCII characters
color	a color name or an RGB triplet representing the color - The syntax of an RGB triplet is: [<red>, <green>, <blue>]. For example, the triplet for green is: ["0", "255", "0"]. - See Appendix B: 'Color Table' on page 300 for a list of supported colors. The list contains the color names and their RGB triplets.
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

```
     Tip: UFDL is case sensitive. All option
          names are lowercase.

The following syntax notation conventions have been used in the
sections following:
 - Names have been assigned to each expression on the right hand
   side of the assignment operator (=). The meaning and setting of
   each expression appear in a table below the syntax diagram.
   For example,
    fontinfo = [<font name>, <point size>, <weight>, <effects>,
              <form>];
    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

 - In the table, the Setting column indicates whether the
   expression requires variable data or a constant value. Variable
   data is represented by a data type; a constant value is
   represented by the required keyword. Data types appear in
   italics (for example, string); constants display in bold face
   (for example, underline).
 - A set of mutually exclusive choices is represented by a list of
   settings beside an expression's name in the table. For example,
   in the fontinfo statement, the <weight> expression can be one
 - The syntax of an expression can take many forms. For example,
   the following formats are all valid:
    value = "Sample expression";
    value = field_one.value;
    value = "Sample " +. field_two.value;

As a consequence of the variation, syntax diagrams make no
reference to an expression's format.

See the section 'Option Definition' for a discussion of expression
formats.
 - Repeating expressions are represented using an <opt1>, ...
  <optn> notation and an explanatory note. For example,
  datagroup = [<datagroup reference1>, ... <datagroup referencen>];
  Note: Include a <datagroup reference> entry for each datagroup
        this item accesses.

 - Optional expressions are noted in an explanatory note. For example,
   fontinfo = [<font name>, <point size>, <weight>,
              <effects>, <form>];
```

Note: <weight>, <effects> and <form> are optional.

5.1 activated

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

Syntax

```
activated = "<status>";
```

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

Available In

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

Example

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

```
saveButton = new button
{
   type = "save";
   value = "Save";
   bgcolor = [activated=="on" ? "white" : "LightPink3"];
}
```

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

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 UFDL 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 - comoboxes 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.

5.2 active

The active option specifies whether an item is active or inactive. Inactive items do not respond to user input and, if possible, appear dimmed.

For example, an inactive check box will be dimmed and the user will not be able to select or deselect the box.

Syntax

```
active = <status>;
```

Expression	Setting	Description
<status>	"on"	item is active
	"off"	item is inactive

Available In
- action
- button

- cell
- check
- field
- help
- label
- list
- popup
- radio
- tablet

Example

This sample specifies the item is active.
active = "on";

Usage Notes

1) Default: on

2) Setting active to off would be similar to setting an edit state of readonly.

5.3 bgcolor

The bgcolor option defines the background color of a page or an item.

Syntax

bgcolor = [<color name>];
bgcolor = [<RGB triplet>];
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 syntax of an RGB triplet.

Available In

- button
- check
- field
- list
- popup
- radio
- tablet
- toolbar
- page global characteristics
- form global characteristics

Examples

These samples both set the background color to forest green.

```
  bgcolor = ["forest green"];
  bgcolor = ["34", "139", "34"];
  bgcolor = ["transparent"];

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
------
```

5.4 bordercolor

The bordercolor option defines the color of the border around the item.

Syntax

```
  bordercolor = [<color name>];
  bordercolor = [<RGB triplet>];
  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 section 5 for the syntax of an RGB triplet.

Available In

- box
- button
- check
- field
- list
- popup
- radio
- tablet
- page global characteristics
- form global characteristics

Examples

```
  bordercolor = ["light blue"];
  bordercolor = ["173", "216", "230"];
```

Usage Notes

1) Default: black

5.5 borderwidth

The borderwidth option defines the width of an item's border. The unit of measurement is pixels.

Syntax

```
borderwidth = <width>;
```

Expression	Setting	Description
<width>	short int	the width of the border

Available In

- box
- button
- field
- label
- list
- popup
- tablet
- page global characteristics
- form global characteristics

Example

This sample sets the border width to five pixels.

```
borderwidth = "5";
```

Usage Notes

1) Default: varies depending on the item type
   - box and label items: zero pixels
   - all other visible items: one pixel

5.6 coordinates

The coordinates option 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.

Syntax

```
 coordinates = [<X_coordinate>, <Y_coordinate>];
```

Expression	Setting	Description
<X_coordinate>	coordinate	the coordinate on the X axis

<Y_coordinate>	coordinate	the coordinate on the Y axis

Available In

- button

Example

When a user clicks on a button containing an image, a coordinates option statement is inserted into the button 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 = ["180", "255"];
```

Usage Notes

5.7 data

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

Syntax

```
data = <data_item>;
```

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

Available In

- action
- button
- cell

Example

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

```
encloseImageButton = new button
{
   value = "Update Image";
   type = "enclose";
   data = "displayImage";
}
```

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

to in the data option.)

Usage Notes

1) A data option may specify 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, labels, and tablets 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 may also have a datagroup option and thus belong to the datagroups of other actions, buttons, or cells.

5.8 datagroup

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

Syntax

```
datagroup = [<datagroup reference1>, ... <datagroup referencen>];
where <datagroup reference> is one of:
- <datagroup name> for datagroups on the current page
- <page tag>.<datagroup name> for datagroups on other pages
Note: Include a <datagroup reference> entry for each datagroup
      this item accesses.
```

Expression	Setting	Description
<datagroup reference>	string	identifies a datagroup

Available In

- action
- button
- cell
- data

Example

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

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

```
datagroup = ["Business_Letters", "Personal_Letters",
             "Form_Letters"];
```

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.

5.9 delay

The delay option delays the execution of an automatic action or specifies an automatic action repeat factor. Repeated actions stop when the page containing the action definition closes. Define automatic actions using an action item.

Syntax

```
delay = [<repeat factor>, <interval>];
```

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 performing single occurrence actions. The unit of measurement is seconds.
	"-1"	perform the action before the page displays. Only valid with a repeat factor of once.

Available In

- action

Example

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

```
  delay = ["once", "900"];
```

Usage Notes

1) Defaults:
- repeat factor: once
- interval: zero seconds
This means the action will occur when the page appears.

2) 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.

3) All actions with the same interval occur in the order they are defined in the page.

4) The page does not display while actions with an interval of -1 are running.

5.10 editstate

The editstate option defines one of three possible edit states for modifiable items.

Syntax

```
editstate = <edit state>;
```

Expression	Setting	Description
<edit state>	"readonly"	users cannot change the item's
	"writeonly"	users can change, but not see, the item's setting
	"readwrite"	users can see and change the item's setting

Available In

- check
- field
- list
- popup
- radio

Example

This sample sets the editstate to readonly.

```
editstate = "readonly";
```

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 may not update the item's contents.

5.11 filename

The filename option identifies the name of an enclosed file. This name appears in the list of enclosed files.

Syntax

```
filename = <file name>;
```

Expression	Setting	Description
<file name>	string	the name of the enclosed file

Available In

- data

Example

This sample specifies the name of an enclosed file.

```
filename = "std_logo.frm";
```

Usage Notes

1) Default: none

2) To ensure cross-platform compatibility, you should 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, you should limit form names to a maximum of eight characters, followed by a .frm extension.

5.12 focused

The focused option specifies whether an item, page, or form currently has the input focus. This option is set by code outside UFDL.

Syntax

```
focused = "<status>";
```

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

Available In
- button
- combo

- field
- list
- popup
- radio
- page global
- form global

Example

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

```
saveButton = new button
{
   type = "save";
   value = "Save";
   bgcolor = [focused=="on" ? "white" : "blue"];
}
```

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 UFDL 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.

5.13 fontcolor

The fontcolor option 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 tablet items, fontcolor defines the pen color. In other items, it defines the text color.

Syntax

```
fontcolor = [<color name>];
fontcolor = [<RGB triplet>];
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 section 5 for the syntax of an RGB triplet.

Available In

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

Examples

These samples both set the background color to chocolate.

```
fontcolor = ["210", "105", "30"];
```

Usage Notes

1) Default: black

5.14 fontinfo

The fontinfo option 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.

Syntax

```
fontinfo = [<font name>, <point size>, <weight>, <effects>,
            <form>];
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 a plain face
```

	"bold"	use a bold face
<effects>	"underline"	underline the text
<form>	"italic"	use the italic form

Available In

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

Example

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

```
fontinfo = ["Times", "14", "bold", "italic"];
```

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) UFDL supports the following fonts and font sizes:
   Fonts: Courier, Times, Symbol (??????), Helvetica, and Palatino
   Sizes: 8, 9, 10, 11, 12, 14, 16, 18, 24, 36, 48

   You can also use other fonts and font sizes if you wish. However, especially for cross-platform Internet applications, it is best to choose from the ones cited above since they are guaranteed to work.

5.15 format

The format option allows you to specify edit checks and formatting

options for field, label, list, popup, and combobox items. It also allows you to specify a mandatory status for signature button items (for details, see the button item description).

Syntax

```
format = [<data type>, <format flag>, <check flag>];
Notes:
i) Multiple flags are valid.
ii) <data type> is mandatory and must appear first; the flags are
    optional and can appear in any order.
```

Expression	Setting	Description
<data type>	(see below)	the type of data the field should contain
<format flag>	(see below)	the type of formatting applied to the user's input
<check flag>	(see below)	the type of edit check performed on the formatted input

Available In

- button
- combobox
- field
- label
- list
- popup

Example

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 = ["integer", "comma_delimit", range=["10", "1000"]];
```

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

```
format = ["dollar", "mandatory", message= "Entry incorrect-try
         again."];
```

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 = ["date", "long"];
```

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

```
format = ["string", template=["###-###-####",
         "###-###-####-###"] ];
```

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 item is mandatory and the user must complete it.

```
format = ["string", page1.allowIncompleteCheck=="on" ? "optional"
```

```
            : "mandatory"];
```

Data Types

UFDL supports the following data types:

Data Type	Description	Format Defaults To:
string	free form character data up to 32K long	Any Character.
integer	a positive or negative whole number in the range of -2,147,	Any whole number.
float	a positive or negative floating point decimal number in the range of 1.7 * 10-308 to 1.7 * 10308	Any decimal number.
dollar	a fixed point decimal number with a scale of 2 and a range equal to the range of a float	Any number. Automatically adds .00 to end, if no decimal value specified
date	a date including day-of-month, month, and year	This format: 3 Mar 96
day_of_week	the name or number of a day of the week	This format: Thu
month	the name or number of a month	This format: Mar
day_of_month	the number of a day of the month	Number format.
year	a numeric year designation	This format: 1996 | 2000 B.C.
time	a time value containing hours and minutes from the 12 hour or the 24 hour clock	This format: 11:23 P.M.
void	disable entire format option (including data type, checks, and formats)	No effects on contents of a field

Format Flags

You can specify any number of format flags in a format line. 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

comma_delimit	Delimit the thousands by commas.

space_delimit	Delimit the thousands by spaces.
bracket_negative	Indicate negative values by surrounding the value with parentheses, that is ( ).
add_ds	Add a dollar sign to the start of the value (dollar fields only).
upper_case	Convert alphabetic characters to upper case.
lower_case	Convert alphabetic characters to lower case.
title_case	Convert first letter of each word to upper case and all other letters to lower case, for titles and proper names.
short	Display dates and times using the following formats: - day_of_week - numeric value in range 1 to 7 where 1 represents Sunday - day_of_month - numeric value in range 1 to 31 - year - apostrophe followed by last two digits in year ('98), 'before Christ' era designator is B.C. ('98 B.C.) - date - year as four digits, month as two digits, and day-of-month as two digits, organized in YMD order; no punctuation (1998-04-29) - time - 24 hour clock (as in 23:30)
long	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 designator is B.C. (2000 B.C.) - date - long year, long month, and long day-of-month formats organized in DMY order; no punctuation (29th April 1998) - time - 12 hour clock with the time of day suffix (A.M. or P.M., as in 11:30 P.M.)
numeric	Display dates and times using numeric values and, possibly, the minus sign:

- day_of_week - 2 digits in range 01 to 07 where 01 represents Sunday
- day_of_month - 2 digits in range 01 to 31

	- month - 2 digits in range 01 to 12 - year - 4 digits; 'before Christ' era designator is minus sign as in -1995 - date - month and day-of-month formats above, * year format is 4 digits * '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"	Available only when formatting dates, to create custom template for presentation 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
void	No formatting is applied

Check Flag

You can specify any number of edit checks in a format line. The edit checks you specify 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: UFDL specifies that fields be formatted before an edit check is performed. For example, if the field's data type is dollar and you specify the add_ds and comma_delimit format options, 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
optional	Input from the user is not mandatory.
mandatory	Input from the user is mandatory.
case_insensitive	Apply edit checks without regard to the case in which the user enters the data.

range=["low","high"]	The field's value must be in the range specified. The range can be alphabetic, numeric, days of the week, days of the month, or months.

	Ranges cannot vary from high to low. For example, 10 to one, the year 2000 to 1900, etc. are invalid.
length=["min","max"]	Restrict the length of the formatted input data to a minimum of "min" bytes and a maximum of "max" bytes.
template=["a","b",...]	This is a list of formats permitted for the field. There is no restriction on the number of formats. Field contents must match one of the formats in the list. You may use any of the following wild card characters: -?- represents any one (1) character? -*- represents any number of characters -#- represents any one (1) numeric character -%- represents any number of numeric characters -@- represents any one (1) alphabetic character -!- represents any number of alphabetic characters (which can include none)
message="help"	Sets the error message that is displayed if the user input fails the type checking. The default message is, "This entry is invalid. Please try again."
fail_checks	Forces failure of format statement.
ignore_checks	Causes all type checking checks to be ignored. Note: only checks are ignored, not formatting or data type.

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

Data Type	Applicable Format Flags	Applicable Check Flags
string	lower_case, upper_case, title_case	case_insensitive, fail_checks, length, mandatory, optional, range, template
integer	bracket_negative, comma_delimit, space_delimit	fail_checks, ignore_checks, length, mandatory, optional, range, template
float	bracket_negative,	fail_checks,

comma_delimit, space_delimit ignore_checks, length,
mandatory, optional,
range, template

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

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 may result in a field the user cannot change. For example, the combination of data type integer and check flag template="a*" 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.

5.16 group

The group option 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.

Syntax

```
group = <group reference>;
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

Available In

- cell
- combobox
- list
- popup
- radio

Example

This sample associates the item with the group coverage_type.

```
group = "coverage_type";
```

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.

5.17 help

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

Syntax

```
help = <item reference>;
```

Expression	Setting	Description
<item reference>	string	identifies the help item

Available In

- button
- check
- field
- label
- list
- popup
- radio
- tablet

Example

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

```
help = "page_1.general_help";
```

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

Syntax

```
image = <item reference>;
```

Expression	Setting	Description
<item reference>	string	identifies the data item

Available In

- button

- label
- tablet

Example

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

```
image = "page_lst.company_logo";
```

Usage Notes

1) Default: none

2) Use this option to associate images with button, label, and tablet items.

3) If an enclosure mechanism is used to replace an image stored in a data item with a new image, then buttons, labels, and tablets 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.

5.19 itemlocation

The itemlocation option serves three purposes:

- It specifies the location of an item in the page layout.
- If you use the extent setting, it will set the size of an item's bounding box.
- If you are using the relational positioning scheme, it may 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 you can use 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. Once you understand the two schemes you can combine them to gain the advantages of both schemes.

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

Note: You can combine the two methods of positioning, so that some items are positioned absolutely, and other items are positioned in relation to those absolute items.

Syntax:

```
itemlocation = [[<specification1>], ... [<specificationn>]];
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 may be negative only when the modifier is offset.

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

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

a) For the Absolute Positioning Scheme:

Modifier	Description

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

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

b) 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 item.
alignb2c	Align bottom edge of item with vertical center of reference point item.
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 item.

alignc2l Align horizontal center of item with left

	edge of reference point item.
alignc2r	Align horizontal center of item with right edge of reference point item.
alignc2t	Align vertical center of item with top
alignhorizbetween	Align horizontal center of item between right edge of first reference point item and left edge of second reference point item.
alignhorizc2c	Align horizontal center of item with horizontal center of reference point item; center below.
alignl2c	Align left edge of item with horizontal center of reference point item.
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 item.
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 point item and top edge of second reference point item.
alignvertc2c	Align vertical center of item with vertical center of reference point item.

Expansion Modifiers (Relational Positioning only)

Modifier	Description

expandb2c	Hold top edge of item constant and expand bottom edge to align with vertical center of reference point item.
expandb2t	Hold top edge of item constant and expand bottom edge to align with top edge of reference point item.
expandl2c	Hold right edge of item constant and expand left edge to align with horizontal center of reference point item.
expandl2l	Hold right edge of item constant and expand left edge to align with left edge of reference point item.
expandl2r	Hold right edge of item constant and expand left edge to align with right edge of reference point item.
expandr2c	Hold left edge of item constant and expand right edge to align with horizontal center of reference point item.
expandr2l	Hold left edge of item constant and expand right edge to align with left edge of reference point item.
expandr2r	Hold left edge of item constant and expand right edge to align with right edge of reference point item.
expandt2b	Hold bottom edge of item constant and expand top edge to align with bottom edge of reference point item.
expandt2c	Hold bottom edge of item constant and expand top edge to align with vertical center of reference point item.
expandt2t	Hold bottom edge of item constant and expand top edge to align with top edge of reference point item.

The Extent Modifier (Relational and Absolute Positioning)

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

Available In

```
          - box
          - button
          - check
          - field
          - label
          - line
          - list
          - popup
          - radio
          - spacer
          - tablet
```

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 itemlocation line describing the label in the picture above would look like this:

```
 itemlocation = [["absolute", "60", "60"]];
```

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

You can offset an item from either:

```
    - Its original absolute position. For example,
    itemlocation = [["absolute", "60", "100"],
         ["offset", "15", "20"]];
    - Its original relational position. For example,
    itemlocation = [["below", "persInfo_label"],
         ["offset", "15", "20"]];
```

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 UFDL 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) may change from one platform to another as the font it is measured in changes size.

If you rely on spacing items on your form using absolute positioning, which fastens an item to a particular pixel, some items may appear overlapped on some platforms.

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

Relational Positioning Scheme

Relational positioning allows you to place an item relative to the location of another item. It also allows you to specify 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 you use the relational positioning scheme, the first external item you place 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 you do not specify any relational position for an item, 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 you to set an absolute size for an item in pixels. When you specify an extent, the item's top left corner will stay where it is, and the item will be resized so that it is as many pixels wide as the x value and as many pixels in height as the y value.

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

Examples

Example 1 - 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.

```
 persInfo_label = new label
 {
    value = "Personal Information";
    itemlocation = [["absolute", "20", "30"]];
 }
```

Example 2 - Offsetting an Item

These samples show two ways in which to offset a field below the label in example one. The first sample shows how to do so using

only the absolute positioning scheme. The second sample shows how to do so using both relational and absolute positioning schemes.

```
 lastName_field = new field
 {
 label = "Last Name";
 itemlocation = [["absolute", "20", "100"],
 }


 lastName_field = new field
 {
 label = "Last Name";
 itemlocation = [["below", "persInfo_label"],
     ["offset", "10", "10"]];
 }
```

Note that the item is offset from its original position, not from other items. It's not a good idea to offset items using strictly absolute positioning (sample one). Use relational positioning if possible (sample two). For more information on the dangers of absolute positioning, see the Caution in section 5.19.

Example 3 - Relational Positioning

Sample 3.1

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 = [["alignvertbetween", "label_one", "label_two"]];
```

Sample 3.2

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 = [["alignl2c", "the_firm"], ["expandr2r",
               "the_firm"]];
```

Sample 3.3

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

```
 itemlocation = [["within", "main_toolbar"], ["below",
               "company_logo"]];
```

Example 4 - Extent

Sample 4.1

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 = [["absolute", "10", "10"], ["extent", "300",
```

"30"]];

Sample 4.2

The second sample shows an extent setting on a label that has been placed using relational positioning. The label is first placed below a field, and is then sized to be 100 pixels wide and 20 pixels high.

```
itemlocation = [["below", "field_1"], ["extent", "100", "20"]];
```

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

Default bounding box size:
See 'Appendix B: Default Sizes'

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) See the following sections for more information on using itemlocation:
   - 'Item Placement'
   - 'Item Size'

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 for more information.

5.20 justify

The justify option aligns lines of text within the space an item occupies.

Syntax

```
justify = <alignment>;
```

Expression	Setting	Description
<alignment>	"left"	align each line's left edge along

"right"	align each line's right edge along the right margin
"center"	align the center of each line with

the center of the item

Available In
- button
- field
- label
- popup
- tablet

Example

This sample aligns the text in the center of the item.
justify = "center";

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

Usage Notes

1) Default: varies depending on the item
    - button and popup items: center
    - label items: left

5.21 label

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

Syntax

label = <label text>;

Expression	Setting	Description
<label text>	string	the text of the label

Available In
- check
- field
- list
- popup
- radio
- spacer
- page global characteristics
- form global characteristics

Example

This sample defines a typical label.

```
     label = "Student Registration Form";

   Usage Notes

    1) Default: none

    2) The label you define in a label option has a transparent
       background by default. If you wish to display a particular color
       behind the label, then set the labelbgcolor option.

    3) Multiple line labels require line breaks imbedded in the label
       text. Use the escape sequence '\n' to indicate a line break.
       For example:
         label = "This label spans\ntwo lines.";
------

  5.22   labelbgcolor

    The labelbgcolor option defines the background color for the label
    specified in the label option.
   Syntax

     labelbgcolor = [<color name>];
     labelbgcolor = [<RGB triplet>];
     Note: Either format is acceptable.

    Expression     Setting        Description
    <color name>   color          the color name
    <RGB triplet>  color          the RGB triplet. See 'Data Type
                                  Designators' on page 116 for the
                                  syntax of an RGB triplet.

   Available In
               - check
               - field
               - list
               - radio
               - page global characteristics
               - form global characteristics

   Examples

    These samples both set the background color to red.
     labelbgcolor = ["red"];
     labelbgcolor = ["255", "0", "0"];
     labelbgcolor = ["transparent"];

   Usage Notes

    1) Default for version 4.0.1 and greater forms: transparent
       This means that a label option will always be transparent unless
       you specify a color.
```

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.

5.23 labelbordercolor

The labelbordercolor option defines the color of the border around the label specified in the label option.

Syntax

```
labelbordercolor = [<color name>];
labelbordercolor = [<RGB triplet>];
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 section 5 for the syntax of an RGB triplet.

Available In

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

Examples

These samples both set the border color to blue1.

```
labelbordercolor = ["blue1"];
labelbordercolor = ["0", "0", "255"];
```

Usage Notes

1) Default: black

5.24 labelborderwidth

The labelborderwidth option defines the width of the border around the label specified in the label option. The unit of measurement is pixels.

Syntax

```
labelborderwidth = <width>;
```

Expression	Setting	Description
<width>	short int	the width of the border

Available In

- check
- field

- list
- radio
- page global characteristics
- form global characteristics

Example

This sample sets the border width to 15 pixels.

```
labelborderwidth = "15";
```

Usage Notes

1) Default: zero pixels

5.25 labelfontcolor

The labelfontcolor option defines the font color for the label specified in the label option.

Syntax

```
labelfontcolor = [<color name>];
labelfontcolor = [<RGB triplet>];
```

Note: Either format is acceptable.

Expression	Setting	Description
<color name>	color	the color name
<RGB triplet>	color	the RGB triplet. See 'Data Type Designators' on page 116 for the syntax of an RGB triplet.

Available In

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

Examples

```
labelfontcolor = ["green1"];
labelfontcolor = ["0", "255", "0"];
```

Usage Notes

1) Default: black

5.26 labelfontinfo

The labelfontinfo option defines the font name, point size, and

font characteristics for the label specified in the label option.

Syntax

labelfontinfo = [<font name>, <point size>, <weight>, <effects>, <form>];
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

Available In

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

Example

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

```
labelfontinfo = ["Palatino", "12", "underline"];
```

Usage Notes

1) See the section on fontinfo for the usage notes.

5.27 mimedata

The mimedata option 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. A UFDL generator should 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.

Syntax

```
 mimedata = <data>;
```

Expression	Setting	Description
<data>	string	the binary data or enclosed file contents

Available In

- data
- signature

Example

This sample assigns some encoded data to the mimedata option. Notice the quotation marks surrounding each segment of the data.

```
 mimedata =
 "R0lGODdhYABPAPAAAP///wAAACwAAAAAYABPAAAC/4SPqcvtD02Y"
 "Art68+Y7im7ku2KkzXnOzh9v7qNw+k+TbDoLFTvCSPzMrS2YzmTE+";
```

This sample shows a mimedata option in a digital signature.

```
 empSignature = new signature
   {
      signformat = "application/uwi_form";
      signer = "Jane D Smith, jsmith@insurance.com";
      signature = "PAGE1.empSignature";
      signitemrefs = ["omit", "PAGE1.mgrSigButton",
             PAGE1.admSigButton",
             "PAGE1.empSignature", "PAGE1.mgrSignature",
             "PAGE1.admSignature"];
      signoptions = ["omit", "triggeritem", "coordinates"];
      mimedata = "MIIFMgYJKoZIhvcNACooIIFIzCCBR8CAQExDzANBgkg"
        "AQUFADALB\ngkqhkiG9w0BBwGgggQZMCA36gAwSRiADjdhfHJl"
        "6hMrc5DySSP+X5j\nANfBGSOI\n9w0BAQQwDwYDVQQHEwhJbn"
        "Rlcm5ldDEXMBUGA1UEChM\nOVmVyaVNpZ24sIEluYy4xNDAKn"
        "1ZlcmlTaWduIENsYXNzIDEgQ0Eg\nLSJbmRdWFsIFN1YnNjcmliy"
        "ZXIwHhcNOTgwMTI3MwMDAwOTgwM\M1OTU5WjCCARExETA";
   }
```

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 to the signer's key be previously

installed.

5.28 mimetype

The mimetype option defines the MIME type of the data stored in a data item.

Syntax

```
mimetype = <MIME type>;
```

Expression	Setting	Description
<MIME type>	string	the MIME type of the data item

Available In

- data

Example

This sample sets the MIME type to indicate image data.

```
mimetype = "image/gif";
```

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). You can find them on the World Wide Web.

MIME type	Meaning
application/postscript	Binary item
application/uwi_bin	Binary item
application/uwi_form	UFDL 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

5.29 mouseover

The mouseover option specifies whether the mouse pointer is currently over an item or page. This option is set by code outside UFDL.

Syntax

```
mouseover = "<status>";
```

Expression	Setting	Description
<status>	"on"	mouse pointer is over item or page
	"off"	mouse pointer is not over item or page

Available In

- button
- check
- combo
- field
- list
- popup
- tablet
- toolbar
- page settings

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.

```
saveButton = new button
{
   type = "save";
   value = "Save";
   bgcolor = [mouseover=="on" ? "white" : "blue"];
}
```

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 toobar).

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

5.30 next

The next option 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.

See the section 'Defining Tabbing and Paging' in section 2.4o for more information on tabbing.

Syntax

```
next = <item reference>;
```

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

Available In

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

Example

This sample points to the item address_field. When users tab ahead from the current item, the item identified as address_field will receive focus.

```
next = "address_field";
```

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 you define them

When you tab past the last item on the page, the first modifiable item in the page's toolbar receives focus. If there is no toolbar, focus returns to the first item.

2) 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.

3) 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

---

5.31 previous

The previous option 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.

See the section 'Defining Tabbing and Paging' in section 2.4o for more information on tabbing.

```
previous = <item reference>;
```

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

Available In

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

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";
```

Usage Notes

2) 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 you define them

   When you tab 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.

4) 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.

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.

5.32 printsettings

The printsettings option determines the settings that will be used when the form is printed. You can allow the user to change these defaults, or set the form so that it will always follow the defaults.

Syntax

printsettings = [<page list>, <dialog settings>];
Notes:
i) All settings are optional.

Expression	Setting	Description
<page list>	(see below)	the list of pages that should be printed
<dialog settings>	(see below)	determines whether the print dialog is shown, and which settings should be used when printing (for example, paper orientation and number of copies)

Available In

- action
- button
- cell
- page global characteristics
- form global characteristics

Page List

The page list uses the following syntax:
pages=["keep"|"omit", "<page tag 1>", "<page tag 2>", ...]

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 should be either kept or omitted.

Dialog Settings

The dialog settings use the following syntax

dialog=[active="on"|"off", orientation="portrait"|"landscape", copies="1"]

The settings work as follows:

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

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=[pages=["omit", "page2"], dialog=["on",
              orientation="landscape", copies="2"]];
```

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.

5.33 saveformat

The saveformat option specifies what format a form should be saved in. A UFDL form can be saved in UFDL format, compressed UFDL format, or HTML format.

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

Compressed UFDL format saves the entire form description as a compressed file using a gzip compression algorithm.

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, radio.

Syntax

```
saveformat = <MIME type>;
```

Expression	Setting	Description
<MIME_type>	"application/uwi_form"	use UFDL format

```
                "application/uwi_form;content-      use compressed
                              encoding=\"gzip\""    UFDL format

                "application/x-www-form-urlencoded" use HTML format


 Note: You cannot specify that HTML format files be compressed.

Available In
           - button
           - cell
Example

 Example 1 - HTML format in a button

 This example shows how to use saveformat in a save button.
  save_button = new button
  {
     type = "save";
     saveformat = "application/x-www-form-urlencoded";
  }

 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.

 Example 2 - Compressed UFDL format in form global characteristics

 This example shows how to use saveformat as a form global
 characteristic and to specify that the saved form be compressed.
  version = "3.2.0";
  bgcolor = ["ivory"];
  saveformat = "application/uwi_form; content-encoding = \"gzip\"";

  page_1 = new page
  {
  ...

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

 A saveformat setting as a form global characteristic applies to all
 save actions for the form. You can override the global setting for
 specific save actions by coding a different saveformat option into
 the item that initiates the save action. For example,
  version = "3.2.0";
  bgcolor = ["ivory"];
  saveformat = "application/uwi_form; content-encoding = \"gzip\"";

  page_1 = new page
  {
     save_button = new button
     {
```

```
            type = "save";
            value = "Save Form";
```

```
        saveformat = "application/uwi_form";
    }
```

When the user saves the form by clicking the Save Form button, it will be saved as an uncompressed UFDL form.

Note that the quotation marks around gzip must be escaped.

Usage Notes

1) Default: UFDL format (not compressed)

2) You can include this option 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:
   <item reference>=<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. <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: combo boxes cannot be saved in HTML format.

Substitutions and Omissions:

- Only modifiable items are saved as HTML data. You cannot save a form in HTML format and expect to view it 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.

5.34 scrollhoriz

The scrollhoriz option defines horizontal scrolling options for a field item.

Syntax

```
scrollhoriz = <option>;
```

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

Available In

- field

Example

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

```
scrollhoriz = "always";
```

Usage Notes

1) Default: never

2) The scroll bar displays along the field's bottom edge.

5.35 scrollvert

The scrollvert option defines vertical scrolling options for a field item.

Syntax

```
scrollvert = <option>;
```

<option>	"never"	permit scrolling using the cursor but display no vertical scroll bar
	"always"	permit scrolling and display a vertical scroll bar
	"fixed"	inhibit scrolling and display no

vertical scroll bars

Available In

- field

Example

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

```
scrollvert = "fixed";
```

Usage Notes

1) Default: never

2) The scroll bar displays along the field's right edge.

5.36 signature

The signature option is used in conjunction with the button item to establish the UFDL item name by which a particular digital signature will be identified.

Syntax

```
signature = <name of signature>;
```

Expression	Setting	Description
<name of signature>	string	the name of the signature

Available In

- button
- signature

Example

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

```
signature = "mysig";
```

Usage Notes

1) Default: none

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

3) For more information on filtering, see "Filters"

5.37 signdatagroups

The signdatagroups option 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.

Syntax

```
signdatagroups = [<datagroup filter>, <datagroup reference>,
                  ... <datagroup referencen>];
```

Note: The number of <datagroup reference> entries is optional.

Expression	Setting	Description
<datagroup filter>	"keep"	include datagroups in the <datagroup reference> list with the signature; omit those not in list
	"omit"	omit datagroups in the <datagroup reference> list from the signature; include those not in list
<datagroup reference>	string	identifies a datagroup

Available In

- button
- signature

Example

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

```
signdatagroups = ["keep", "Business_Letters"];
```

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.

3) See "Order of Precedence of Filters" on page 48 for further information on filtering.

5.38 signer

The signer option identifies who signed a particular form.

Syntax

```
signer = <identity of user>;
```

Expression	Setting	Description
<identity of user>	string	identity of user

Available In

- button

- signature

Example

In this example, signer is similar to a user's email signature, clearly identifying who signed the form.

```
signer = "John Smith jsmith@acme.org";
```

Usage Notes

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

5.39 signformat

The signformat option records the type of encoding that a Viewer should 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.

Syntax

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

Expression	Setting	Description
<MIMEtype>	string	the MIME type of the signed data. Must be application/uwi_form.
<csp>	string	the cryptographic service provider

to use. Must be a string enclosed in escaped double-quotation marks. The string is pre-defined by the crypto API.

<csptype>	string or csp-defined number	the type of implementation of the crytographic service provider. For allowed types, see the list in
<alg>	string	the hash algorithm to use

Available In

- button
- signature

Example

```
  empSigButton = new button
  {
     type = "signature";
     value = signer;
     format = ["string", "mandatory"];
     signformat = "application/uwi_form;csp=\"Microsoft Base
  Cryptographic Provider v1.0\";csptype=rsa_full;hashalg=sha1";
     signoptions = ["omit", "triggeritem", "coordinates"];
     signitemrefs = ["omit", "PAGE1.mgrSigButton",
                     "PAGE1.admSigButton", "PAGE1.empSignature",
                     "PAGE1.mgrSignature", "PAGE1.admSignature"];
     signature = "empSignature";
  }
```

Usage Notes

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

2) You must escape the quotation marks around the name of the cryptographic service provider. Escape them using the backslash (\).

3) The list below describes the settings you may use for the csptype parameter. (Note that you may also use a numeric value, 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 Standard
dss_dh	For a CSP that supplies DSS compliant algorithms and Diffie-Hellman encryption
fortezza	For a CSP that supplies Fortezza algorithms

4) Instead of using one of the settings in the table above for csptype, you may use the numeric value that is defined for it in the cryptographic API. For example, csptype=dss and csptype=3

produce the same result.

5.40 signgroups

The signgroups option 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.

Syntax

```
signgroups = [<group filter>, <group reference>, ...
              <group referencen>];
```

Note: The number of <group reference> entries is optional.

Expression	Setting	Description
<group filter>	"keep"	include groups of items in the <group reference> list with the signature; omit those not in list
	"omit"	omit groups of items in the <group reference> list from the signature; include those not in list
<group reference>	string	identifies a group of items

Available In

- button
- signature

Example

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

```
signgroups = ["omit", "yesnoradiobuttons", "monthlypaycells"];
```

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 may be populated by the same group of cells.

3) See "Order of Precedence of Filters" in section 2.7 for further information on filtering.

5.41 signitemrefs

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

Syntax

```
signitemrefs = [<item filter>, <item reference>, ...
                <item referencen>];
```

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; omit those not in list
	"omit"	omit items in the <item reference> list from the signature; include those not in list
<item reference>	string	specifies the item to be filtered

Available In

- button
- signature

Example

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

```
signitemrefs = ["omit", "field1", "page1.field2"];
```

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.

4) See "Order of Precedence of Filters" for further information on filtering.

5.42 signitems

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

Syntax

```
signitems = [<item filter>, <item type>, ... <item typen>];
```

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; omit
	"omit"	omit types of items in the <item type> list from the signature; include those not in list
<item type>	string	specifies the type of item to be

filtered

Available In

- button
- signature

Example

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

```
signitems = ["keep", "box", "button", "field"];
```

Usage Notes

1) Default: keep

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

3) See "Order of Precedence of Filters" in section 2.7 for further information on filtering.

5.43 signoptionrefs

The signoptionrefs option 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.

Syntax

```
signoptionrefs = [<option filter>, <option reference>, ...
                  <option referencen>];
```

Note: The number of <option reference> entries is optional.

Expression	Setting	Description
<option filter>	"keep"	include options in the <option reference> list with the signature; omit those not in list
	"omit"	omit options in the <option reference> list from the signature; include those not in list
<option reference>	string	specifies the option to be filtered

Available In

- button
- signature

Example

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

```
signoptionrefs = ["keep", "page1.field1.value"];
```

Note: the page name can be dropped if the option in question is on the same page, but the item name should not be dropped.

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 UFDL format.

3) Signoptionrefs filters take precedence over signoptions filters.

4) See "Order of Precedence of Filters" in section 2.7 for further information on filtering.

5.44 signoptions

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

Syntax

```
signoptions = [<option filter>, <option type>, ...
               <option typen>];
```

Note: The number of <option type> entries is optional.

<option filter>	"keep"	include types of options in the <option type> list with the signature; omit those not in list
	"omit"	omit types of options in the <option type> list from the signature; include those not in list
<option type>	string	specifies the type of option to be filtered

Available In

- button
- signature

Example

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

```
signoptions = ["omit", "url", "printsettings"];
```

Usage Notes

1) Default: keep

2) One signoptions setting that must always be specified is as

follows:
signoptions = ["omit", "triggeritem", "coordinates"];

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.

4) See "Order of Precedence of Filters" in section 2.7 for further information on filtering.

5.45 size

The size option 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.

See 'Item Size' in section 2.4e for a discussion of item and bounding box sizes.

Syntax

```
size = [<width>, <height>];
```

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

Available In

- box
- button
- check
- field
- label
- line
- list
- popup
- radio
- spacer
- tablet

Example

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

```
size = ["80", "5"];
```

Usage Notes

2) Size and Font:

The width may not always accommodate the number of characters you specify. The calculation to determine actual width is: 'average character width for the item's font' X <width> This will only exactly match the number of characters the item can display horizontally when the font is mono-spaced (like Courier).

3) 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.

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

5.46 thickness

The thickness option specifies the thickness of a line. The unit of measurement is pixels.

Syntax

```
thickness = <thickness>;
```

Expression	Setting	Description
<thickness>	short int	the thickness of the line

Available In

- line

Example

Example 1

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

```
size = ["40", "0"];
thickness = "5";
```

Example 2

This sample defines a vertical line 20 characters long and 10 pixels thick.

```
size = ["0", "20"];
thickness = "10";
```

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 can be changed by using itemlocation with an expansion modifier for the dimension that thickness describes.

5.47 transmitdatagroups

This is one of the transmit-family of options that allow you to filter form submissions. This option lists which datagroups of items should be kept or omitted from a transmission.

For example, if transmitdatagroups specifies that the datagroup called enclosures should be kept in the transmission, then all items with a datagroup setting of enclosures will be transmitted (unless other filters of greater precedence exclude them).

This filter applies only to items for which it is valid to have a datagroup option (action, button, cell, data).

For details on the order of precedence of filters, see "Order of Precedence of Filters" in section 2.7.

Syntax

```
transmitdatagroups = [<transmit flag>, <datagroup identifier1>,
                  ... <datagroup identifiern>];
```

Note: There may be zero or more <datagroup identifier> entries.

Expression	Setting	Description
<transmit flag>	"keep"	keep items with a datagroup setting specified in the <datagroup identifier> list; omit those with a datagroup setting not included in the list
	"omit"	omit items with a datagroup setting specified in the <datagroup identifier> list; keep those with a group setting not in the list
<datagroup identifier>	string	the name of a datagroup setting

Available In

- button
- cell

Examples

This sample specifies that only the items with a datagroup setting of enclosures should be transmitted.

```
transmitdatagroups = ["keep", "enclosures"];
```

This sample specifies that all items except those with a datagroup

```
    setting of other should be kept in the transmission.
     transmitdatagroups = ["omit", "other"];
```

Usage Notes

1) The default is to keep all datagroups in the form.

For details on the order of precedence of filters, see "Order of Precedence of Filters" in section 2.7.

5.48 transmitformat

The transmitformat option specifies the format of the form data submitted to a processing application. A UFDL form can submit data in UFDL format, compressed UFDL format, or in HTML format.

UFDL format is the entire form definition, including user input, unless you use the options transmit, transmititems, and transmitoptions to omit some items and options from the form submission.

HTML format is 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.

If you specify that a UFDL format submission should be compressed, it will be compressed using a gzip compression algorithm.

Note: Form and page global characteristics are sent only if the format is UFDL.

Syntax

```
transmitformat = <MIME_type>;
```

Expression	Setting	Description
<MIME_type>	"application/uwi_form"	use UFDL format
	"application/uwi_form;content-encoding=\"gzip\""	use compressed UFDL format
	"application/x-www-form-urlencoded"	use HTML form format

Note: You cannot specify that data in HTML form format be compressed.

Available In

- action
- button
- cell
- form global characteristics

Examples

Example 1 - UFDL format

This example shows a button which, when clicked, will submit the

```
form in UFDL format.
 send_button = new button
 {
    type = "done";
    url = ["mailto:user@host.domain"];
    transmitformat = "application/uwi_form";
 }
```

When a user clicks the button, the entire form definition will be submitted, unless other transmit options specify a partial submission.

Example 2 - Compressed UFDL format specified in form global

This example shows how to use the transmitformat option as a form global option. Here, it specifies that data should be submitted in compressed UFDL format.

```
 version = "3.2.0";
 bgcolor = ["ivory"];
 transmitformat = "application/uwi_form; content-encoding=\"gzip\"";

 page_1 = new page
 {
```

When a submit or done action is activated in the form, the data will be sent in compressed UFDL format. When transmitformat appears as a form global characteristic, it applies to all submissions from the form. You can override it for a particular submission if you place a transmitformat setting in the item that initiates the submission.

Note that the quotation marks around gzip must be escaped.

Example 3 - HTML form format

This sample shows an automatic action that submits form data in HTML form format.

```
 status_action = new action
 {
    type = "submit";
    url = ["http://www.host.domain/cgi-bin/recvStatus"];
    transmitformat = "application/x-www-form-urlencoded";
    delay = ["repeat", "180"];
 }
```

Every 180 seconds, the form definition will be converted to HTML form format as specified in Usage Note 4. Other transmit options could override the choice of items to include in an HTML form (see Usage Notes 5 and 6).

Usage Notes

1) Default: UFDL format (not compressed)

2) You can include this option 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: combo boxes 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) Partial Submissions

Just as you can specify partial submissions when transmitting
data in UFDL format, you can also specify partial submissions
when transmitting data in HTML format. Use the transmit and
transmititems options.

Use the transmitoptions option for HTML formatted submissions
with caution. If you omit the options used for HTML format, then
items requiring those options are omitted also.
For example, if the trigger item's definition included the

following pair of statements, the form submission would contain only radio item entries (all other entries use a value option setting).

```
 transmitformat = "application/x-www-form-urlencoded";
```

```
        transmitoptions = ["omit", "value"];

    5) HTML Considerations

       The functionality of UFDL forms differs somewhat from HTML
       forms. Those differences are:
        - Enclosures
          HTML does not support enclosures. To submit enclosed form
          data, use UFDL format.
        - Item tags
          UFDL allows a smaller set of characters in item tags than
          HTML does. UFDL item tags support the following characters:
          a-z, A-Z, 0-9, and the underscore ( _ ).
        - Check boxes
          UFDL check boxes vary slightly from HTML check boxes. UFDL
          check boxes are independent items; HTML check boxes are
          grouped together using the same format as radio items. When
          a UFDL form is submitted in HTML format, the submission will
          contain an entry for each check box.
------

  5.49   transmitgroups

    This is one of the transmit-family of options that allow you to
    filter form submissions. This option lists which groups of items
    should be kept or omitted from a transmission.

    For example, if transmitgroups specifies that the group called
    countries should be kept in the transmission, then all items with
    a group setting of countries will be transmitted (unless other
    filters of greater precedence exclude them).

    This filter applies only to items for which it is valid to have a
    group option (cell, combobox, list, popup, radio).

    For details on the order of precedence of filters, see "Order of
    Precedence of Filters"

   Syntax

     transmitgroups = [<transmit flag>, <group identifier1>,
                  ... <group identifiern>];
     Note: There may be zero or more <group identifier> entries.

    Expression         Setting   Description
    <transmit flag>    "keep"    keep items with a group setting
                                 specified in the <group identifier>
                                 list; omit those with a group setting
                                 not included in the list
                       "omit"    omit items with a group setting
                                 specified in the <group identifier>
                                 list; keep those with a group setting
```

		not in the list
<group identifier>	string	the name of a group setting

Available In

- action
- button
- cell

Examples

This sample specifies that only the items in the countries and departments groups should be kept in the transmission.

```
transmitgroups = ["keep", "countries", "departments"];
```

This sample specifies that all groups should be kept in the transmission except the fillType group.

```
transmitgroups = ["omit", "fillType"];
```

Usage Notes

1) The default is to keep all groups in the form.

2) This option is handy for keeping the user's selection in lists and popups. Since the value setting of a popup or list is not the value of the cell the user chose, but is rather the item tag of the cell containing the value, you might want to make sure you keep the selected cell in the transmission so that you can dynamic option reference its value. To do this, you would keep the group of cells associated with the popup's group.

   For details on the order of precedence of filters, see "Order of Precedence of Filters"

5.50 transmititemrefs

This is one of the transmit-family of options that allow you to filter form submissions. This option lists which specific items should be kept or omitted from a transmission. It differs from transmititems in that transmititems specifies particular types of items to filter, whereas transmititemrefs refers to one or more specific items.

For example, if transmititemrefs specifies that the item page1.MgrSignButton should be kept in the transmission, then even if a transmititems filter says all buttons should be omitted from the form, the particular button on page 1 named MgrSignButton would be kept.

For details on the order of precedence of filters, see "Order of Precedence of Filters"

Syntax

```
transmititemrefs = [<transmit flag>, <item identifier1>,
```

... <item identifiern>];

Note: There may be zero or more <item identifier> entries.

Expression	Setting	Description

<transmit flag>	"keep"	keep the specific items referred to in the <item identifier> list; omit all other items
	"omit"	omit the specific items referred to in the <item identifier> list; keep all others
<item identifier>	string	the item reference of an item, starting with the page tag

Available In

- action
- button
- cell

Examples

This sample specifies that only the item on page 1 called MgrSignButton should be transmitted, and that all other items should be omitted.

```
transmititemrefs = ["keep", "page1.MgrSignButton"];
```

This sample shows how you would use transmititemrefs in conjunction with transmititems: although all items that are buttons are omitted, the button on page 1 called MgrSignButton will be kept.

```
transmititems = ["omit", "button"];
transmititemrefs = ["keep", "page1.MgrSignButton"];
```

Usage Notes

2) The default is to keep all items in the form.

3) The setting of a transmititemrefs always overrides a transmititems setting.

4) The setting of a transmitoptionrefs always overrides a transmititemrefs setting.

For full details on the order of precedence of filters, see "Order of Precedence of filters" in section 2.7.

5.51 transmititems

This option lists the types of items to include in or omit from the form data submitted to a form processing application. Include this option in the definitions of items that trigger form submissions. These trigger items have a type option setting of submit or done.

Syntax

```
transmititems = [<transmit flag>, <item type1>, ... <item typen>];
```

Note: The number of <item type> entries is optional.

Expression	Setting	Description
<transmit flag>	"keep"	include items with an item type from

		the <item type> list; omit those not in list
	"omit"	omit items with an item type from the <item type> list; include those not in list
<item type>	string	a type of item

Available In

- action
- button
- cell

Example

This sample specifies that box, help, label, spacer, and toolbar items should be omitted from the form data submitted to the form processing application.

```
transmititems = ["omit", "box", "help", "spacer", "toolbar"];
```

Usage Notes

1) The default is to keep all items.

2) See the transmititemsrefs description for information on how to keep or omit a specific item (as opposed to a type of item).

5.52 transmitoptionrefs

This is one of the transmit-family of options that allow you to filter form submissions. This option lists which specific options should be kept or omitted from a transmission. It differs from transmitoptions in that transmitoptions specifies particular types of options to filter, whereas transmitoptionrefs refers to one or more specific options.

For example, if transmitoptionrefs specifies that the option page1.NameField.value should be kept in the transmission, then even if a transmitoptions filter says all value options should be omitted from the form, the particular value option in the field on page 1 called NameField will be kept.

For details on the order of precedence of filters, see "Order of Precedence of Filters" in section 2.7.

Syntax

```
transmitoptionrefs = [<transmit flag>, <option identifier1>,
                  ... <option identifiern>];
```

Note: There may be zero or more <option identifier> entries.

Expression	Setting	Description
<transmit flag>	"keep"	keep the specific options referred to

in the <option identifier> list; omit
all other options

"omit" omit the specific options referred to

	in the <option identifier> list; keep all others
<option identifier> string	the option reference of an option, starting with the page tag

Available In

- action
- button
- cell

Examples

This sample shows how you would use transmitoptionrefs in conjunction with transmitoptions: although all options that are values are omitted, the value in the NameField on page 1 will be kept.

```
transmitoptions = ["omit", "value"];
transmitoptionrefs = ["keep", "page1.NameField.value"];
```

This sample shows how you would use transmitoptionrefs in conjunction with transmititemrefs: although the item called MgrSignButton on page 1 is omitted, its signer option is kept

```
transmititemrefs = ["omit", "MgrSignButton"];
transmitoptionrefs = ["keep", "page1.MgrSignButton.signature"];
```

Usage Notes

5) The default is to keep all options in the form.

For details on the order of precedence of filters, see "Order of Precedence of Filters" in section 2.7.

5.53 transmitoptions

This option lists which options to include in or omit from the form data submitted to a form processing application. Include this option in the definitions of items that trigger form submissions. These trigger items have a type option setting of submit or done.

Only options meeting the following standard are affected by transmitoptions:

- The option definition occurs in an item already included in the form submission.

Note: The version option is always included in the submission unless the format is HTML.

Syntax

```
transmitoptions = [<transmit flag>, <option identifier1>,
```

... <option identifiern>];

Note: The number of <option identifier> entries is optional.

Expression	Setting	Description

<transmit flag>	"keep"	include options with an option type in the <option identifier> list; omit those not in list
	"omit"	omit options with an option identifier in the <option identifier> list; include those not in list

Tip: Transmitoptions does not affect options in items omitted by the transmititems option.

Available In

- action
- button
- cell

Example

This sample specifies that only the active, mimedata, and value options should be included in the form data submitted to the form processing application.

```
transmitoptions = ["keep", "active", "mimedata", "value"];
```

Usage Notes

6) The default is to keep items.

7) See the transmitoptionrefs description for details on how to keep or omit a specific option (as opposed to a type of option).

5.54 triggeritem

The triggeritem option 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.

Syntax

```
triggeritem = <item reference>;
```

Expression	Setting	Description
<item reference>	string	identifies the trigger item

Available In

- form global characteristics

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";
------
```

5.55 type

The type option associates a task type with an item that can trigger a task: action, button, or cell.

Syntax

```
     type = <task type>;
```

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

Task Types

The <task type> can be any of the following:

Task Type	Description of Task	Use With
link	Perform all requests specified by the url options in the current item. See the section 'url' for more details.	action button cell
replace	Perform a link followed by a cancel.	action cell
submit	Initiate the form processing applications identified in the url options of the current item.	action button cell
done	Perform a submit followed by a cancel.	action button cell
pagedone	Move to the page specified in the url option. This closes the current page and replaces it with the new page. All fields containing error checking on the current page must be correctly filled out before it can be closed.	action button cell
cancel	Close the form; if any changes were made to the form since the last save or submit, then the user is informed that the form has changed and is allowed to choose whether the cancellation should proceed. Note that the value options of many items, as well as the contents of data items, can change in response to an enclose	action button cell

or remove action.

save Save the form in a file specified by action

	the user.	button cell
select	With cell items: flag the cell as selected when a user chooses the cell. This means the item reference of the cell is copied to the value option of the parent list or popup. With button items containing images: store coordinates of the mouse click position in the image into the coordinates option	button cell
enclose	Allows the user to place one or more files into one or more of the datagroups defined for the form. The files will be encoded using base64 encoding format.	button cell
extract	Allows a user to copy the contents of an enclosed file into a file on the local disk.	button cell
remove	Allows the user to remove an item from a datagroup; the underlying data item will only be deleted if it belongs to no other datagroups.	button cell
display	Display an enclosed file. Your web browser will choose the appropriate viewer according to the file's MIME type.	action button cell
print	Print the form on a local printer.	action button cell
signature	Create a digital signature.	button

Available In

- action
- button
- cell

Example

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

```
type = "save";
```

Usage Notes

1) Default: link

5.56 url

The url option 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 can be any 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 UFDL 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.

Syntax

```
url = [<the URL1>, <the URL2>, ... <the URLn>];
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

Available In

- action
- button
- cell

Example

This sample identifies a form processing application.

```
url = ["http://www.host.domain/cgi-bin/recv_status"];
```

```
    This sample identifies a page to display and an item on it to
    direct the focus to.
     url = ["#page_2.expense_field"];
```

Usage Notes

1) Default: none

2) When a form communicates with a server, the information sent may 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 triplet
\r	13	%0D

Applications receiving form data must check the content type of the incoming data to see whether it is url-encoded.

5.57 value

The value option 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).

Syntax

```
value = <setting>;
```

Expression	Setting	Description
<setting>	string	the item's contents

Available In

- button
- cell
- check
- field
- help
- label
- list
- popup
- radio
- tablet

Example

This sample identifies the text of a label item.
value = "My Form Title";

Usage Notes

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

2) Multiple line values require line breaks imbedded in the value text. Use the escape sequence '\n' to indicate a line break. For example:
value = "This value spans\ntwo lines.";

3) To get the value of a cell that a user has selected from a list, you need to dereference it, like this:
<page_tag>.<list_tag>.value->value
For example:
page1.countryPopup.value->value

When a user selects a cell from a list, the item tag of the cell is stored as the value of the list. Hence the dereference syntax.

5.58 version

The version option specifies the Universal Forms Description Language version used to code the form definition.

Note: This option must be the first statement in the form. It is invalid in any other location.

Syntax

version = <version>;

Expression	Setting	Description
<version>	string	the UFDL version number.

Note: The version number must be coded as a quoted string. Other expression formats are invalid in this statement.

Available In

- form global characteristics (mandatory)

This sample indicates the language in this form definition conforms to UFDL version 3.2.0 specification.
version = "3.2.0";

Usage Notes

1) Important: Do not increase the version number of old forms

unless they have been modified to conform to the new version.

2) The format of a version number is m.n.r where:

i) m is the major version number
ii) n is the minor version number
iii) r is the maintenance release number

5.59 <custom option>

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

Syntax

<custom> = [<expression1>, ... <expressionn>];
Note: <expression> can occur 1-n times.

Expression	Setting	Description
<expression>	string	an expression that assigns a value to the option

Example

This sample shows a custom option containing an SQL query.

```
sql_query = ["SELECT NAME FROM EMPLOYEE WHERE ID = "];
```

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.

Usage Notes

1) The naming conventions for a custom option are as follows:
   - It must begin with an alphabetic character.
   - It can contain any of the characters A-Z, a-z, 0-9, $ and underscore.
   - It must contain an underscore.

---

6. UFDL Form Viewer Directive

UFDL contains the following viewer directive statement:
- the #include statement for including external files
- the #optinclude statement for optionally including external files

The syntax of a viewer directive statement is as follows:

```
#<directive> "<value>"
```

Do not terminate viewer directive statements with a semicolon.

See the section "UFDL Form Options" for syntax notation conventions in section 5.0.

6.1 #include

The #include statement allows you to include external files in your form definition much as you would include header files in a C language source file. The form viewer application replaces the #include statement with the contents of the include file before the form appears.

Syntax

```
#include "<filename>"
```

Value	Setting	Description
<filename>	string	name of the include file

Example

This is an example of using an include file to add image data to a form.

```
?
// Create a label to display the image.
LOGO_IMAGE = new label
{
  image = "LOGO_DATA";
}
//
// Now include the image in the form.
#include "logo.frm"
?
```

This is the external file:

```
LOGO_DATA = new data
{
 mimedata = "<image data>";
}
```

The form after resolution of the include:

```
?
// Create a label to display the image.
LOGO_IMAGE = new label
{
  image = "LOGO_DATA";
}
//
// Now include the image in the form.

LOGO_DATA = new data
```

```
{
 mimedata = "<image data>";
}
```

```
     ?
```

Usage Notes

1) You can code a #include statement anywhere in a form definition except imbedded in another statement. You can also nest #include statements.

2) The include file must reside in a secure include directory accessible to the form viewer application.

3) Use include files to reduce form file size, and ensure standardization of form definitions. Smaller files occupy less space on the server and download more quickly

---

6.2 #optinclude

The #optinclude statement is a variation on the #include statement. It allows you to include external files in your form definition much as you would include header files in a C language source file-without crashing your program if a file you attempt to include is not available. The form viewer application replaces the #optinclude statement with the contents of the optinclude file before the form appears.

Syntax

```
#optinclude "<filename>"
```

Value	Setting	Description
<filename>	string	name of the optinclude file

Example

This is an example of using an optinclude file to add image data to a form.

Here is the original form definition:

```
?
// Create a label to display the image.
{
  image = "LOGO_DATA";
}
//
// Now include the image in the form.
#optinclude "logo.frm"
?
```

This is the external file:

```
     LOGO_DATA = new data
     {
```

```
 mimedata = "<image data>";
}


The form after resolution of the include:
 ?
 // Create a label to display the image.
 LOGO_IMAGE = new label
 {
   image = "LOGO_DATA";
 }
 //
 // Now include the image in the form.

 LOGO_DATA = new data
 {
  mimedata = "<image data>";
 }

 ?


Usage Notes

 1) Employing the same syntax as #include, #optinclude is a
    convenient alternative to #include, because:

    - In a given pool of users, everyone can be sent the same form,
      but certain users can have access to all its components while
      others do not-but there is no risk of crashing for anyone.

    - #optinclude saves server resources by making decisions on the
       client side about which files are to be included.

 2) The files to be included must reside in a secure directory on
    the user's computer.
------
```

7. UFDL Functions

UFDL functions allow forms to perform procedural logic, and also to perform complex operations that would normally require complicated conditional statements to achieve.

Using functions in conjunction with conditional statements and user events provides a means for creating extremely powerful Internet form applications in a fairly simple and elegant manner.

Below is a simple example of using a UFDL function (in the status_option line):

```
 version = "4.0.0";
```

```
     page1 = new page
     {
        field1 = new field
```

```
    {
            label = "Field 1";
            format = ["string", "mandatory"];
            value = "high";
    }

    {
            label = "Field 2";
            format = ["string", "mandatory"];
            status_option = toggle(field1.value, "high", "low");
            value = status_option=="1"?"Declined":"";
    }

 }
```

Explanation: The toggle function (used in the status_option option line) returns a "1" if the value in field1 changes from high to low. As a result, the value of field 2 will change to Declined, if the toggle function returns a "1". Otherwise the value of the second field will be empty.

Function Definition

Call a UFDL function much in the same way you would call a C or Java function. The formal definition of a UFDL function is:

```
 returnValue package.functionName([parameter])
```

Expression	Description
returnValue	a string or an array
package	the name of the package that the function belongs to; UFDL functions from this specification are part of the system package. All custom-made packages must contain an underscore in their names.
functionName	the name of the function
parameter	a string or an array

Notes on the Functions

Position in Strings

When referring to the position of a character in a string, 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

1) To pass a literal value into a function, surround it in
   double-quotatation marks. For example:

```
        str_length = strlen("This is a literal string");

    2) 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 = strlen(surnameField.value);

    3) To pass an option reference into a function (without evaluating
       it), surround the option reference in double-quotation marks.
       For example:
        auto_set = set("statusField.value", "Confirmed.");

    A UFDL form will evaluate option references before a function runs,
    unless the references are surrounded by quotation marks.

   About Functions and Packages

    Functions are compiled into packages, which must reside on the
    desktop computer. The UFDL functions that are documented in this
    specification must be compiled into a package called system. No
    functions other than those documented here may be part of the
    system package.

    Form developers are free to develop their own packages of functions
    to extend UFDL. Packages of custom-developed functions must contain
    an underscore in their names. For example:

    - extr_fun.ifx

   About UFDL Functions

    UFDL functions are divided into the following sections:

     - String Functions
     - Math Functions
     - Utility Functions
     - Time and Date Functions
------

 7.1   String Functions

  7.1a   countLines

   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 you need to dynamically size items into
    which you want to insert a string. For example, if you want to
```

insert an entry from a database into a field on a form, you can 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.

Call

```
countLines(<string>, <width>)
```

Parameters

Expression	Setting	Description
<string>	literal string or option reference	the string to base the measurement on
<width>	positive int	the width, in monospaced characters, to base the measurement on

Returns

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

Example

```
commentField = new field
{
   label = "Comments";
   itemlocation = [["below", "deptField"]];
   size = ["50", countLines(value, "50")];
}
```

In the example above, the field's height will be set by the number that countLines returns. The calculation is based on a dynamically-generated value, and the field's set width (50).

7.1b replace

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.

Call

```
  replace(<string>, <start>, <end>, <newString>)
```

Parameters

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

Returns

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

Example

```
  replaceField = new field
  {
     label = "Test replace()";
     format = ["string", "mandatory"];
     id_value = replace(value, "3", "6", "east");
  }
```

The result of replace in the above example would be "Go east, young man!".

7.1c strlen

Description

Returns the length of <string>.

Call

```
  strlen(<string>)
```

Parameters

Expression	Setting	Description

<string>	literal string or option	the string (enclose literal strings in double quotation marks, do not

	reference	enclose option references in quotation marks)

Returns

A string containing the length.

Example

```
stringLengthField = new field
{
   label = "The length of this label is:";
   format = ["string", "mandatory"];
   value = strlen(label);
}
```

The result of strlen in the above example would be "28".

7.1d strmatch

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.

Call

```
strmatch(<wild>, <real>)
```

Parameters

Expression	Setting	Description
<wild>	literal string or option reference	the wildcard string to match (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
<real>	literal string or option reference	the non-wildcard match string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

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

Example

```
     testStrmatch = new field
     {
        label = "Test strmatch()";
```

```
        format = ["string", "mandatory"];
        value = "To be or not to be, etc.";
        id_value = strmatch("?o be* ?o be*", value);
     }
```

The result of strmatch in the above example would be "1".

7.1e strpbrk

Description

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

Call

```
strpbrk(<string1>, <string2>)
```

Parameters

Expression	Setting	Description
<string1>	literal string or option reference	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
<string2>	literal string or option reference	the string of characters (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

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

Example

```
testStrpbrk = new field
{
   label = "testField";
   format = ["string", "mandatory"];
   value = "To be or not to be, etc.";
   id_value = strpbrk(value, "lLmMnNoOpP");
}
```

The result of strpbrk in the above example would be "9".

7.1f strrstr

Description

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

Call

```
strrstr(<string1>, <string2>)
```

Parameters

Expression	Setting	Description
<string1>	literal string or option reference	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
<string2>	literal string or option reference	the substring (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

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

Example

```
testStrrstr = new field
{
   label = "testField";
   format = ["string", "mandatory"];
   value = "To be or not to be, etc.";
   id_value = strrstr(value, "be");
}
```

The result of strrstr in the above example would be "16".

7.1g strstr

Description

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

Call

```
strstr(<string1>, <string2>)
```

Parameters

Expression	Setting	Description
<string1>	literal string or option reference	the string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)
<string2>	literal string	the substring (enclose literal

or option reference	strings in double quotation marks, do not enclose option references

in quotation marks)

Returns

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

Example

```
testStrstr = new field
{
   label = "testField";
   format = ["string", "mandatory"];
   value = "To be or not to be, etc.";
   id_value = strstr(value, "be");
}
```

The result of strstr in the above example would be "3".

7.1h substr

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.

Call

```
substr(<string>, <start>, <end>)
```

Parameters

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

Returns

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

Example

```
surnameField = new field
{
   label = "Surname";
   format = ["string", "mandatory"];
   value = "Watson";
   id_value = substr(value, "0", "4");
}
```

The result of substr in the above example would be "Watso".

7.1i tolower

Description

Returns the lower case of <string>.

Call

```
tolower(<string>)
```

Parameters

Expression	Setting	Description
<string>	literal string or option reference	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The lower case string.

Example

```
tolowerField = new field
{
   label = "Type in Here";
   format = ["string", "mandatory"];
   value = "Hello!";
   id_value = tolower(value);
}
```

The result of tolower in the above example would be "hello!".

7.1j toupper

Description

Returns the upper case of <string>.

Call

```
toupper(<string>)
```

Parameters

Expression	Setting	Description
<string>	literal string or option reference	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The upper case string.

Example

```
toupperField = new field
{
   label = "Type in Here";
   format = ["string", "mandatory"];
   value = "Hello!";
   id_value = toupper(value);
}
```

The result of toupper in the above example would be "HELLO!".

7.1k trim

Description

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

Call

```
trim(<string>)
```

Parameters

<string>	literal string or option reference	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The string with leading and trailing whitespace removed.

```
     trimField = new field
     {
        label = "     Test trim()  ";
        format = ["string", "mandatory"];
        value = trim(label);
     }
```

The result of trim in the above example would be "Test trim()".

7.1l URLDecode

Description

Returns a URL-decoded version of <string>.

Call

```
URLDecode(<string>)
```

Parameters

Expression	Setting	Description
<string>	literal string or option reference	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The URL-decoded string.

Example

```
     URLDecodeField = new field
     {
        label = " Test URLDecode";
        format = ["string", "mandatory"];
        value = URLDecode("This%20is%20a%20line%0D");
     }
```

The result of URLDecode in the above example would be "This is a line\r".

7.1m URLEncode

Description

Returns a URL-encoded version of <string>.

Call

URLEncode(<string>)

Parameters

Expression	Setting	Description
<string>	literal string or option reference	the original string (enclose literal strings in double quotation marks, do not enclose option references in quotation marks)

Returns

The URL-encoded string.

Example

```
URLEncodeField = new field
{
   label = " Test URLEncode";
   format = ["string", "mandatory"];
}
```

The result of URLEncode in the above example would be "This%20is%20a%20line%0D".

7.2 Math Functions

7.2a abs

Description

Returns the absolute value of the number represented in <number>. An error occurs if <number> is not a valid number.

Call

```
abs(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

Example

```
absTest = new field
```

```
    {
       label = "Test abs";
```

```
        format = ["string", "mandatory"];
        value = abs("-2341.23");
     }
```

The result of abs in the above example would be "2341.23".

7.2b acos

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.

Call

```
acos(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

Example

```
arccosineField = new field
{
   label = " Test acos";
   format = ["string", "mandatory"];
   value = acos("0.5");
}
```

7.2c annuity

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>.)

You might use this function to figure out either:

- P, the present value (lump sum to invest)
- R, the periodic payment amount that you will receive

For your reference:

P = R * annuity_factor R = P / annuity_factor

An error occurs if <periods> is not a valid integer, or if <rate> is 0.

Call

```
annuity(<rate>, <periods>)
```

Parameters

Expression	Setting	Description
<rate>	decimal number	the rate of interest in decimal form compounded each period
<periods>	integer	the number of periods

Returns

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

Example

```
presentValueInv = new field
{
   label = "The present value to invest is:";
   format = ["string", "mandatory"];
   value = paymentField.value * annuity(".05", "7");
}
```

In the example above, annuity would return "5.786373", and, if the desired payment entered into paymentField were $1, then the value of presentValueInv would be $5.78. (That is, a person would have to invest $5.78 at 5% for seven payments.)

7.2d asin

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.

Call

```
asin(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

Example

```
arcsineField = new field
{
   label = " Test asin";
   value = asin("0.5");
}
```

The result of asin in the above example would be "0.523599".

7.2e atan

Description

Returns the arc tangent of a number stored in <number>.

An error occurs if <number> is not a valid number.

Call

```
atan(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

Example

```
arctangentField = new field
{
   label = " Test atan";
   format = ["string", "mandatory"];
   value = atan("0.5");
}
```

The result of atan in the above example would be "0.463648".

7.2f ceiling

Description

Returns the ceiling of the number represented in <number>.

An error occurs if <number> is not a valid number.

Call

```
ceiling(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

Example

```
ceilingTest = new field
{
   label = "Test ceiling";
   format = ["string", "mandatory"];
   value = ceiling("-19.6");
}
```

The result of ceiling in the above example would be "-19".

7.2g compound

Description

Returns the compound interest factor at a rate indicated by <rate> over a number of periods specified in <periods>.

You might use this to calculate the total amount of a loan, by multiplying an original principle by the result of compund. See below for an example.
An error occurs if <periods> is not a valid integer.

Call

```
compound(<rate>, <periods>)
```

Parameters

Expression	Setting	Description
<rate>	decimal number	the rate of interest in decimal form compounded each period

<periods>	integer	the number of periods

Returns

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

Example

```
totalAmountField = new field
{
   label = "Total Amount of Loan";
   format = ["string", "mandatory"];
   value = principleField.value * compound(".1", "7");
}
```

The result of compound in the above example would be "1.948717". The value of the field would then be 1.948717 x the amount in the field called principleField.

7.2h cos

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.

Call

```
cos(<angle>)
```

Parameters

Expression	Setting	Description
<angle>	decimal number	the angle in radians

Returns

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

Example

```
cosineField = new field
{
   label = " Test cos";
   format = ["string", "mandatory"];
   value = cos("2");
}
```

The result of cos in the above example would be "-0.416147".

7.2i deg2rad

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.

Call

```
deg2rad(<angle>)
```

Parameters

Expression	Setting	Description
<angle>	decimal number	the angle in degrees

Returns

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

Example

```
deg2radField = new field
{
   label = " Test deg2rad";
   format = ["string", "mandatory"];
   value = deg2rad("114.591559");
}
```

The result of deg2rad in the above example would be "2.00000".

7.2j exp

Description

Returns the exponentiation of the number represented in <number> (i.e., e<number>).

An error occurs if <number> is not a valid number.

Call

```
exp(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

error occurs.

Example

```
expTest = new field
{
   label = "Test exp";
   format = ["string", "mandatory"];
   value = exp("3");
}
```

The result of exp in the above example would be "1.098612".

7.2k fact

Description

Returns the factorial value of the integer represented in <integer>.

An error occurs if <integer> is negative.

Call

```
fact(<number>)
```

Parameters

Expression	Setting	Description
<integer>	integer	a non-negative integer

Returns

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

Example

```
factTest = new field
{
   label = "Test fact";
   format = ["string", "mandatory"];
   value = fact("8");
}
```

The result of fact in the above example would be "40320".

7.2l floor

Description

Returns the floor of the number represented in <number>.

An error occurs if <number> is not a valid number.

Call

```
floor(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

Example

```
floorTest = new field
{
   label = "Test floor";
   format = ["string", "mandatory"];
   value = floor("-19.6");
}
```

The result of floor in the above example would be "-20".

7.2m ln

Description

Returns the natural logarithm of the number represented in <number>.

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

Call

```
ln(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number

Returns

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

Example

```
     lnTest = new field
     {
        label = "Test ln";
        format = ["string", "mandatory"];
        value = ln("1");
     }
```

The result of ln in the above example would be "0".

7.2n log

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.

Call

```
log(<number>)
log(<number>, <base>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number
<base>	decimal number	a number representing the base for which the logarithm will be computed

Returns

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

Example

```
     logTest = new field
     {
        label = "Test log";
        format = ["string", "mandatory"];
        value = log("100", "10");
     }
```

The result of log in the above example would be "2".

7.2o mod

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.

Call

```
mod(<number>, <divisor>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number
<divisor>	decimal number	a number representing the divisor for which the modulus will be computed

Returns

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

Example

```
modTest = new field
{
   label = "Test mod";
   format = ["string", "mandatory"];
   value = mod("-3.5", ".3");
}
```

The result of mod in the above example would be "-0.200000".

7.2p pi

Description

Returns the value of PI to the best available accuracy.

Call

```
pi()
```

Parameters

Expression	Setting	Description
(none)		

Returns

A string containing the value of PI.

Example

```
     piTest = new field
     {
        label = "Test pi";
        format = ["string", "mandatory"];
        value = pi();
     }
```

The result of pi in the above example would be "3.14159265359" (precision is machine-dependent).

7.2q power

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.

Call

```
power(<number>, <power>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a number
<power>	decimal number	a number representing the power by which the number will be raised

Returns

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

Example

```
     powerTest = new field
     {
        label = "Test power";
        format = ["string", "mandatory"];
        value = power("0.1", "-2");
     }
```

The result of power in the above example would be "100.00000".

7.2r rad2deg

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.

Call

```
rad2deg(<angle>)
```

Parameters

Expression	Setting	Description
<angle>	decimal number	the angle in radians

Returns

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

Example

```
rad2degField = new field
{
   label = " Test rad2deg";
   format = ["string", "mandatory"];
   value = rad2deg("2");
}
```

The result of rad2deg in the above example would be "114.591559".

7.2s rand

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>.

Call

```
rand(<lowerlimit>, <upperlimit>)
```

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

Returns

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

Example

```
randTest = new field
{
   label = "Test rand";
   format = ["string", "mandatory"];
   value = rand("45", "90");
}
```

The result of rand in the above example would be an integer in the range [45,90].

7.2t round

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.

Call

```
round(<number>, <place>)
```

Parameters

Expression	Setting	Description
<place>	decimal number	a number representing the decimal place where <number> is to be rounded

Returns

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

Example

```
roundTest = new field
{
   label = "Test round";
   format = ["string", "mandatory"];
   value = round("-323.235", ".01");
}
```

The result of round in the above example would be "-323.2400".

7.2u sin

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.

Call

```
sin(<angle>)
```

Parameters

Expression	Setting	Description
<angle>	decimal number	the angle in radians

Returns

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

Example

```
sineField = new field
{
   label = " Test sin";
   format = ["string", "mandatory"];
   value = sin("2");
}
```

The result of sin in the above example would be "0.909297".

7.2v sqrt

Description

Returns the square root of the number represented in <number>.

An error occurs if <number> is a negative number.

Call

```
sqrt(<number>)
```

Parameters

Expression	Setting	Description
<number>	decimal number	a non-negative number

Returns

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

Example

```
    {
       label = "Test sqrt";
       format = ["string", "mandatory"];
       value = sqrt("19.5");
    }
```

The result of sqrt in the above example would be "4.415880".

7.2w tan

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

Call

```
tan(<angle>)
```

Parameters

Expression	Setting	Description
<angle>	decimal number	the angle in radians

Returns

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

Example

```
tanField = new field
{
   label = " Test tan";
   format = ["string", "mandatory"];
   value = tan("2");
}
```

The result of tan in the above example would be "-2.185040".

7.3 Utility Functions

7.3a applicationName

Description

Returns the name of the currently running application.

Call

```
     applicationName()

   Parameters

    Expression     Setting          Description
    (none)

   Returns

    A string containing the application name.

   Example

     anTest = new field
     {
        label = "Test applicationName";
        value = applicationName();
     }

    The result of applicationName in the above example, if the form
    were running in an application named "Viewer", would be "Viewer".
------

  7.3b   applicationVersion

   Description

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

   Call

     applicationVersion()

   Parameters

    Expression     Setting          Description
    (none)

   Returns

    A string containing the application version.

   Example

     avTest = new field
     {
        label = "Test applicationVersion";
        format = ["string", "mandatory"];
        value = applicationVersion();
     }
```

The result of applicationVersion in the above example, if running in an application at version 3.2.4, would be "03.02.04".

7.3c applicationVersionNum

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.

Call

```
applicationVersionNum()
```

Parameters

Expression	Setting	Description
(none)		

Returns

A string containing the application version number.

Example

```
avnTest = new field
{
   label = "Test applicationVersionNum";
   format = ["string", "mandatory"];
   value = applicationVersionNum();
}
```

The result of applicationVersionNum in the above example, if running in an application at version v3.2.4, would be "50463744", which is the decimal representation of 0x03020400.

7.3d decimal

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>.

Call

```
decimal(<number>, <base>)
```

Parameters

Expression	Setting	Description
<number>	number	a number
<base>	positive integer	an integer that is the base of the provided number

Returns

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

Example

```
decimalTest = new field
{
   label = "Test decimal";
   format = ["string", "mandatory"];
   value = decimal("-4a", "16");
}
```

The result of decimal in the above example would be "-74".

7.3e formatString

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.

Call

```
formatString(<string>, <itemtagOfFormat>)
```

Parameters

Expression	Setting	Description
<string>	a string	a string to format according to the referenced option
<formatOptionReference>	an option reference, including the page tag, if necessary	the option reference of the format line to use when in formatting the string

Returns

The formatted string.

```
   Example
     Field1 = new field
     {
```

```
      label = "Field 1";
      format = ["dollar", "add_ds", "comma_delimit"];
      value = "";
    }
    Field2 = new field
    {
      label = "Field 2";
      format = ["string", "mandatory"];
      value = formatString(Field3.value, "Field1.format");
    }
    Field3 = new field
    {
      label = "Field 3";
    ]
```

The result of formatString in Field2 would be $30,095.60.

```
    //Example 2
    Field4 = new field
    {
      value = "$1.00";
      format = ["dollar"];
      backend_value = formatString(value,"backend_format");
      backend_format = ["integer"];
      }
```

In the example above, formatString is used to reformat a value as an integer and insert it into a custom option (presumably for a back-end application to use).

7.3f isValidFormat

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.

Call

```
isValidFormat(<string>, <formatOptionReference>)
```

Parameters

Expression	Setting	Description
<string>	a string	a string to be checked against the format
<formatOptionReference>	an option reference,	the option reference of the format to check the string

	including the page tag, if necessary	against

Returns

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

Example

```
Field1 = new field
{
   label = "Test isValidFormat1";
   format = ["integer", "mandatory"];
   value = "45";
}
Field2 = new field
{
   label = "Test isValidFormat2";
   format = ["string", "mandatory"];
   value = isValidFormat("23.2", "Field1.format");
}
```

The result of isValidFormat in the above example would be "0" because the string to check contains a non-integer number representation and the specified format to check is of type integer.

7.3g set

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.

Call

```
set(<reference>, <value>)
```

Parameters

<reference>	form option reference	an adequately qualified reference to a form option
<value>	form value	a string containing the option's new value

Returns

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

Example

```
field1 = new field
{
   label = "Test set 1";
   format = ["string", "mandatory"];
   value = "gold";
}
field2 = new field
{
   label = "Test set 2";
   format = ["string", "mandatory"];
   value = set("field1.value", "silver");
}
```

The result of set in the above example would be "1" and the value of field1 would be set to "silver".

7.3h toggle

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.

Call

```
toggle(<reference>)
toggle(<reference>, <from>, <to>)
```

Parameters

Expression	Setting	Description
<reference>	form option reference	an adequately qualified reference to a form option
<from>	form value	a string containing a possible option value
<to>	form value	a string containing a possible option value

Returns

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

Example

```
timeStampField = new field
{
   value = toggle(nameField.value)=="1"?now():"";
   label = "Time Stamp";
   editstate = "readonly";
}

nameField = new field
{
   value = "";
}
```

In the example above, toggle has just a <reference> parameter. Every time the nameField's value changes, toggle will return "1", and then a new time will be entered into timeStampField, using the now function.

```
noChoiceAllowed = new check
{
   label = "Simple Application - No Choices";
   value = "off";
   logic_1 = toggle(noChoiceAllowed.value, "off", "on")
                   == "1" ? set("option1.value", "off") +
                   set("option2.value", "off") +
                   set("option3.value", "off") : "";
}
```

In the example above, toggle is used to change form behavior based on whether the noChoiceAllowed check box is checked. When the check's value changes from off to on, toggle will return a "1". That will trigger the decision set up in the logic_1 option - thus the items called option1, option2, and option3 will become deselected (their values will be off). Normally, you would also set their active options to be off, but to save room, this example omits that step.

7.4 Time and Date Functions

7.4a date

Description

Returns the current date in "yyyymmdd" format.

Call

```
date()
```

Parameters

Expression	Setting	Description

(none)

Returns

A string containing the current date.

Example

```
dateTest = new field
{
   label = "Test date";
   format = ["string", "mandatory"];
   value = date();
}
```

The result of date in the above example, if run on January 18th, 1998, would be "19980118".

7.4b dateToSeconds

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.

Call

```
dateToSeconds(<date>, <time>)
```

Parameters

Expression	Setting	Description
<time>	time string	a time in a recognized format

Returns

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

Example

```
dtsTest = new field
{
   label = "Test dts";
   format = ["string", "mandatory"];
   value = dateToSeconds("980319", "09:39:16");
}
```

The result of dateToSeconds in the above example would be "89030056"

7.4c day

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.

Call

```
day(<dateSecs>|"")
```

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
dayTest = new field
{
   label = "Test day";
   format = ["string", "mandatory"];
   value = day("890300356");
}
```

The result of day in the above example would be "19".

7.4d dayOfWeek

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.

Call

dayOfWeek(<dateSecs>|"")

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
{
   label = "Test dayOfWeek";
   format = ["string", "mandatory"];
   value = dayOfWeek("890300356");
}
```

The result of dayOfWeek in the above example would be "5".

7.4e endOfMonth

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.

Call

```
endOfMonth(<dateSecs>|"")
```

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
eomTest = new field
```

```
    {
       label = "Test endOfMonth";
       format = ["string", "mandatory"];
       value = endOfMonth("890300356");
    }
```

The result of endOfMonth in the above example would be "891337156".

7.4f hour

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.

Call

```
hour(<dateSecs>|"")
```

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
hourTest = new field
{
   label = "Test hour";
   format = ["string", "mandatory"];
   value = hour("890300356");
}
```

The result of hour in the above example would be "9".

7.4g minute

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.

Call

```
minute(<dateSecs>|"")
```

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
minuteTest = new field
{
   label = "Test minute";
   format = ["string", "mandatory"];
   value = minute("890300356");
}
```

The result of minute in the above example would be "39".

7.4h month

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.

Call

```
month(<dateSecs>|"")
```

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
monthTest = new field
{
   label = "Test month";
   format = ["string", "mandatory"];
   value = month("890300356");
}
```

The result of month in the above example would be "3".

7.4i now

Description

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

Call

```
now()
```

Parameters

(none)

Returns

A string containing the number of seconds.

Example

```
nowTest = new field
{
   label = "Test now";
   format = ["string", "mandatory"];
   value = now();
}
```

The result of now in the above example, if run at 09:39:16 GMT on Thursday, March 19th, 1998 would be "890300356".

7.4j second

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.

Call

```
second(<dateSecs>|"")
```

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
secondTest = new field
{
   label = "Test second";
   format = ["string", "mandatory"];
   value = second("890300356");
}
```

The result of second in the above example would be "16".

7.4k time

Description

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

Call

```
time()
```

Parameters

Expression	Setting	Description
(none)		

Returns

A string containing the current time.

Example

```
timeTest = new field
{
   format = ["string", "mandatory"];
```

```
        value = time();
    }
```

The result of time in the above example, if run at 3:22 in the afternoon, would be "3:22:PM".

7.4l year

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.

Call

```
year(<dateSecs>|"")
```

Parameters

Expression	Setting	Description
<dateSecs>	number	a date represented by the number of seconds since 00:00:00 GMT, January 1st, 1970

Returns

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

Example

```
yearTest = new field
{
   label = "Test year";
   format = ["string", "mandatory"];
   value = year("890300356");
}
```

The result of year in the above example would be "1998".

Appendix A: Quick Reference Tables

A.1 Table of Items and Form and Page Characteristics

Item	Available Options
action	activated; active; data; datagroup; delay; transmitdatagroups; transmitformat; transmitgroups;

transmititemrefs; transmititems; transmitoptionrefs;
transmitoptions; type; url

box	bgcolor; bordercolor; borderwidth; fontinfo; itemlocation; size
button	activated; active; bgcolor; bordercolor; borderwidth; coordinates; data; datagroup; editstate; focused; fontcolor; fontinfo; format; help; image; itemlocation; justify; mouseover; next; previous; saveformat; signature; signdatagroups; signer; signformat; signgroups; signitemrefs; signitems; signoptionrefs; signoptions; size; transmitdatagroups; transmitformat; transmitgroups; transmititemrefs; transmititems; transmitoptionrefs; transmitoptions; type; url; value
cell	activated; active; data; datagroup; editstate; group; saveformat; transmitdatagroups; transmitformat; transmitgroups; transmititemrefs; transmititems; transmitoptionrefs; transmitoptions; type; url; value
check	active; bgcolor; bordercolor; editstate; focused; fontcolor; fontinfo; help; itemlocation; label; labelbgcolor; labelbordercolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; size; value
combobox	activated; active; bgcolor; bordercolor; borderwidth; editstate; focused; fontcolor; fontinfo; format; group; help; itemlocation; label; labelbgcolor; labelbordercolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; size; value
data	datagroup; filename; mimedata; mimetype
field	active; bgcolor; bordercolor; borderwidth; editstate; focused; fontcolor; fontinfo; format; help; itemlocation; justify; label; labelbgcolor; labelbordercolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; scrollhoriz; scrollvert; size; value
help	active; value
label	active; bgcolor; bordercolor; borderwidth; fontcolor; fontinfo; help; image; itemlocation; justify; size; value
line	fontcolor; fontinfo; itemlocation; size; thickness
list	active; bgcolor; bordercolor; borderwidth;

editstate; focused; fontcolor; fontinfo; group;
help; itemlocation; label; labelbgcolor;
labelbordercolor; labelborderwidth; labelfontcolor;
labelfontinfo; mouseover; next; previous; size;

	value
popup	activated; active; bgcolor; bordercolor; borderwidth; editstate; focused; fontcolor; fontinfo; group; help; itemlocation; justify; label; mouseover; next; previous; size; value
radio	active; bgcolor; bordercolor; editstate; focused; fontcolor; fontinfo; group; help; itemlocation; label; labelbgcolor; labelbordercolor; labelborderwidth; labelfontcolor; labelfontinfo; mouseover; next; previous; size; value
signature	mimedata, signature, signdatagroups, signer, signformat, signitems, signgroups, signoptions, signoptionrefs
spacer	fontinfo; itemlocation; label; size
tablet	active; bgcolor; bordercolor; borderwidth; fontcolor; help; image; itemlocation; justify; mouseover; size; value
toolbar	bgcolor; mouseover
page globals	activated; bgcolor; bordercolor; borderwidth; focused; fontcolor; fontinfo; label; mouseover; next
form globals	activated; bgcolor; bordercolor; borderwidth; focused; fontcolor; fontinfo; label; next; saveformat; transmitformat; triggeritem; version

---

A.2 Table of Options

Option	Details
activated	Syntax: activated = "on" | "maybe" | "off"; Default:off Items: action; button; cell; combobox; popup; page global; form global
active	Syntax: active = "on" | "off"; Default:on Items: action; button; cell; check; combobox; field; help; label; list; popup; radio; tablet. To prevent user input in a field, use the editstate "readonly".
bgcolor	Syntax: bgcolor = ["<color name>"]; bgcolor = ["<R value>","<G value>", "<B value>"];

```
                Default:for button - "gray"
                        for check, field, list, popup, radio -
                        "white" all other items - the background #
                        color of the form
```

```
                Items:  box; button; check; combobox; field; label;
                        list; popup; radio; tablet; toolbar; page
                        characteristics; form characteristics

bordercolor     Syntax: bordercolor = ["<color name>"];
                        bordercolor = ["<R value>","<G value>",
                                       "<B value>"];
                Default:the bordercolor set in the global
                        characteristics, or "black" if no
                        characteristics set
                Items:  box; button; check; combobox; field; label;
                        list; popup; radio; tablet; page
                        characteristics; form characteristics

borderwidth     Syntax: borderwidth = "<width>";
                Default:for label - 0
                        for all other items - the borderwidth set
                        in characteristics, or 1 if no
                        characteristics set
                Items:  box; button; combobox; field; label; list;
                        popup; tablet; page characteristics; form
                        characteristics

coordinates     Syntax: coordinates = ["<X_coordinate>",
                        "<Y_coordinate>"];
                Default:none
                Items:  button

data            Syntax: data = "<data_item>";
                Default:none
                Items:  action; button; cell
datagroup       Syntax: datagroup = ["<datagroup_reference>",
                        "<datagroup reference>"...];
                Default:none
                Items:  action; button; cell; data

delay           Syntax: delay = ["repeat" | "once","interval"];
                Default:once with an interval of 0 seconds
                Items:  action

editstate       Syntax: editstate = "readonly | writeonly |
                        readwrite";
                Default:readwrite
                Items:  button; cell; check; combobox; field; list;
                        popup; radio

filename        Syntax: filename = "<file name>";
                Default:None
                Items:  data

focused         Syntax: focused = "on" | "off";
```

```
Default:off
Items:  button; check; combo; field; list; popup;
        radio; page global; form global
```

```
fontcolor       Syntax: fontcolor = ["<color name>"];
                        fontcolor = ["<R value>","<G value>",
                        "<B value>"];
                Default:for check and radio - red
                        for all other items, the fontcolor set in
                        global characteristics, or "black" if no
                        preference set
                Items:  button; check; combobox; field; label; line;
                        list; popup; radio; tablet; page
                        characteristics; form characteristics

fontinfo        Syntax: fontinfo = ["<font name>","<point size>",
                        "<weight>","<effects>","<form>"];
                        * weight, effects, and form are optional
                Default:the fontinfo set in global characteristics,
                        or "Helvetica 8 plain" if no characteristics
                        set
                Items:  box; button; check; combobox; field; label;
                        line; list; popup; radio; spacer; tablet;
                        page characteristics; form characteristics

format          Syntax: format = [<data type>,<format flag>,
                        <check flag>];
                        * format and check flags are optional, and
                          multiple flags are valid
                Default:for data type - none
                        for format flag - depends on data type
                        for check option - depends on data type
                Items:  button; combobox; field; label; list; popup

group           Syntax: group = "<group name | group reference>";
                Default:none
                Items:  cell; combobox; list; popup; radio

help            Syntax: help = "<item reference>";
                Default:none
                Items:  button; check; combobox; field; label; list;
                        popup; radio; tablet

image           Syntax: image = "<item reference>";
                Default:none
                Items:  button; label; tablet

itemlocation    Syntax: itemlocation = [[<specification>],
                        [<specification>]...];<specification> =
                        "<modifier>","<itemtag>","<itemtag>"
                        * the second itemtag only to align between
                          modifiers
                Default:for the first item - the top left corner of
                        the form
                        for all other items - vertically below the
```

```
                        previously created item and horizontally at
                        the left margin
                Items:  box; button; check; combobox; field; label;
                        line; list; popup; radio; spacer; tablet
justify         Syntax: justify = "<left | right | center>";
                Default:for button and popup - center
                        for label - left
                Items:  button; field; label; popup; tablet

label           Syntax: label = "<label text>";
                Default:none
                Items:  check; combobox; field; list; popup; radio;
                        spacer; page characteristics; form
                        characteristics

labelbgcolor    Syntax: labelbgcolor = ["<color name>"];
                        labelbgcolor = ["<R value>","<G value>",
                        "<B value>"];
                Default:for items in the toolbar - the background
                        color of the toolbar
                        for other items - the background color of
                        the form
                Items:  check; combobox; field; list; radio

labelbordercolorSyntax:labelbordercolor = ["<color name>"];
                        labelbordercolor = ["<R value>","<G value>",
                        "<B value>"];
                Default:black
                Items:  check; combobox; field; list; radio

labelborderwidthSyntax:labelborderwidth = "<width>";
                Default:0 pixels
                Items:  check; combobox; field; list; radio

labelfontcolor Syntax: labelfontcolor = ["<color name>"];
                        labelfontcolor = ["<R value>","<G value>",
                        "<B value>"];
                Default:black
                Items:  check; combobox; field; list; radio; page
                        characteristics; form characteristics

labelfontinfo  Syntax: labelfontinfo = ["<font name>",
                        "<point size>","<weight>","<effects>",
                        "<form>"];
                        * weight, effects, and form are optional
                Default:Helvetica, 8 plain
                Items:  check; combobox; field; list; radio

mimedata        Syntax: mimedata = "<data>";
                Default:none
                Items:  button, data, signature
```

```
    mouseover       Syntax: mouseover = "on" | "off";
                    Default:off
                    Items:  button; check; combo; field; list; popup;
                    radio; tablet; toolbar; page global

    next            Syntax: next = "<item reference>";
                    Default:when the form opens - the first non-toolbar
                            item in the form's description that users
```

```
                        can modify
                        when tabbing to subsequent items - the next
                        item in the form's description that users
                        can modify
                        when tabbing from the last item - the first
                        item in the form's description that users
                        can modify (can be a toolbar item)
                Items:  button; combobox; check; field; list; popup;
                        radio; page globals; form globals

previous        Syntax: previous = "<item_reference>";
                Default:the previous item in the form description
                Items:  button; combobox; check; field; list; popup;
                        radio;

printsettings   Syntax: printsettings = [<page list>,
                        <dialog settings>];
                Default:the page list defaults to include all pages
                        in the form the dialog defaults to "on", has
                        the following settings:
                        orientation = portrait
                        copies = 1
                        printpages active = on
                Items:  action, button, cell, page global
                        characteristics, form global characteristics

saveformat      Syntax: saveformat = "<mimetype>";
                Default:application/uwi_form
                Items:  button; cell; form characteristics

scrollhoriz     Syntax: scrollhoriz = "<never | always | wordwrap>";
                Default:never

scrollvert      Syntax: scrollvert = "<never | always | fixed>";
                Default:never
                Items:  field

signature       Syntax: signature = "<string>";
                Default:none
                Items:button, signature

signdatagroups Syntax: signdatagroups = ["<keep | omit>",
                        "<datagroup reference>","<datagroup
                        reference>"...];
                Default:keep
                Items:  button, signature

signer          Syntax: signer= "<string>";
                Default:depends on where signature is from
                Items:  button, signature
```

```
    signformat     Syntax: signformat = "<MIME type>";
                   Default:application/uwi_form
                   Items:  button, signature
```

```
signgroups      Syntax: signgroups = ["<keep | omit>","<group
                        reference>","<group reference>"...];
                Default:keep
                Items:  button, signature

signitems       Syntax: signitems = ["<keep | omit>", "<item type>",
                        "<item type>"...];
                Default:keep
                Items:  button, signature

signitemrefs    Syntax: signitemrefs = ["<keep | omit>", "<item
                        reference>","<item reference>"...];
                Default:keep
                Items:  button, signature

signoptionrefs Syntax: signoptionrefs = transmitoptions = ["<keep
                        | omit>","<option reference>","<option
                        reference>"...];
                Default:keep
                Items:  button, signature

signoptions     Syntax: signoptions = ["<keep | omit>","<option
                        type>","<option type>"...];
                Default:keep
                Items:  button, signature

size            Syntax: size = ["<width>","<height>"];
                        The unit of measurement is characters.
                Default:see below (defaults are also used in place
                        of invalid and missing arguments)

     Items       Width        Height      Default bounding box size

     box         1            1           same as item
     (smaller than one not allowed in either dimension)

     button
     (with text) label width  label height same as item
     (with image)image width  image height same as item

     check       1            1           max (1, label width)
                                          x  label height  + 1

     combobox    max of       1           same as item
                 (label width,
                 widest cell)

     field       60           1           max (field width, label
                                          width)x field height +
                                          label height

     label
```

(label empty)	1	1	same as item
(label given)	label width	label height	same as item

```
     line         1           1 pixel     same as item
     (one dimension must be 0)


     list         max of       number of  same as item
                  (label       cells, +1
                  width,       if label
                  widest
                  cell)

     popup        max of      1           same as item
                 (label width,
                 widest cell)

     radio        1           1           max (radio width, label
                                          width)x radio height +
                                          label height

     spacer
     (no label)   1           1           same as item
     (label given)label width label heightsame as item

     tablet
     (no image)   1           1           same as item
     (image)      image width image heightsame as item


thickness      Syntax: thickness = "<thickness>";
               Default:1 pixel
               Items:  line
transmitdatagroupsSyntax:transmitdatagroups = [<transmit flag>,
                         <datagroup identifier>, ...
               Default:keep
               Items:  action; button; cell

transmitformat Syntax: transmitformat = "<MIME type>";
               Default:application/uwi_form
               Items:  action; button; cell; form characteristics

transmitgroups Syntax: transmitgroups = [<transmit flag>, <group
                       identifier>,...<group identifier>];
               Default:keep
               Items:  action; button; cell

transmititemrefsSyntax:transmititemrefs = [<transmit flag>, <item
                       identifier>,...<item identifier>];
               Default:keep
               Items:  action; button; cell

transmititems  Syntax: transmititems = ["<keep | omit>", "<item
                       type>","<item type>"...];
               Default:keep
```

Items: action; button; cell

transmitoptionrefsSyntax:transmitoptionrefs = [<transmit flag>,

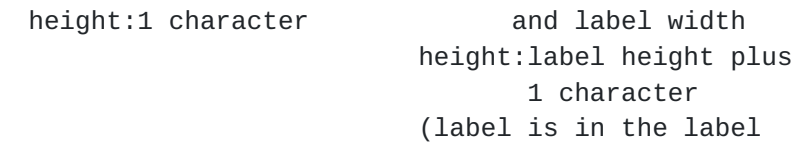
```
                          <option identifier>,...<option
                          identifier>];
                  Default:keep
                  Items:  action; button; cell

    transmitoptionsSyntax: transmitoptions = ["<keep | omit>","<option
                          type>","<option type>"...];
                  Default:keep
                  Items:  action; button; cell

    triggeritem   Syntax: triggeritem = "<item reference>";
                  Default:the item reference of the item that
                          triggered the "submit" or "done"
                  Items:  in form global characteristics

    type          Syntax: type = "<task type>";
                  Default:link
                  Items:  action; button; cell

    url           Syntax: url = ["<URL | item reference>",
                          ["<URL>"],...];
                  Default:none
                  Items:  action; button; cell

    value         Syntax: value = "<setting>";
                  Default:depends on item
                  Items:  button; cell; check; combobox; field; help;
                          label; list; popup; radio; tablet

    version       Syntax: version = <version number>;
                  Default:none
                  Items:in form global characteristics
------
```

Appendix B: Default Sizes

The following table shows the default basic item and bounding box sizes.

box	width:1 character height:1 character Smaller than 1 not allowed in either dimension	Same as default item size
button	width:width of label height:height of label (label is in the value option) or size of embedded image	same as default item size

	height:1 character	and label width height:label height plus 1 character (label is in the label option)
combobox	width:larger of label width and widest cell (iii) height:1 character (label is in the label option)	Same as default item size (ii)
field	width:30 characters height:1 character	width:larger of item width and label width (ii) height:height of item plus height of label (ii) (label is in the label option)
label	width:1 character if label empty, otherwise label width height:1 character if label empty, otherwise label height or size of embedded image if it exists	Same as default item size
line	width:30 character height:1 pixel One dimension must be 0 (i)	Same as default item size
list	width:larger of label width and widest cell (iii) height:number of cells in list (label is in the label	width:larger of item width and widest cell (ii) height:height of item plus height of label (label is in the label option)
popup	width:larger of label width and widest cell (iii) height:1 character (label is in the label option)	Same as default item size (ii)
radio	width:1 character	width:larger of 1 character


height:1 character
and label width
height:label height plus
1 character
(label is in the label

		option)
spacer	width:1 character if label empty, otherwise label width height:1 character if label empty, otherwise label height (label is in the label option)	Same as default item size
tablet	width:1 character if tablet empty, otherwise value width height:1 character if label empty, otherwise value height or size of embedded image if it exists	Same as default item size

Notes

i) For line items, either height or width must be set to zero. The thickness option specifies the thickness (in pixels) of the line in the dimension containing zero (0). If both settings are non-zero, the line size will default to one character wide by one pixel thick.

ii) This includes a scroll bar if one appears.

iii) The cell's width comes from the cell's value option setting.

Appendix C: UFDL for C and C++ Programmers

This document is intended to introduce programmers to the syntax of UFDL. To do so, we will compare UFDL to the C programming language, and point out many of the similarities in syntax and structure that exist between the two languages, as well as some of the differences.

Be aware that this document outlines one way of modelling these similarities and differences, and that a number of other approaches could be used.

C.1 Procedural vs. State Language

Unlike C, UFDL is a state language. Where C describes a procedure that is followed, UFDL describes a state that is maintained. All of

the statements in a UFDL form are always maintained as being true.

As a result, computations are constantly updated throughout the form. For instance, imagine a form with a field called total, the value of which is based on adding the values of five other fields. Whenever a value is entered (or changed) in one of those five fields, the total field will be instantly updated to reflect that change. In this, UFDL acts much like a spreadsheet.

C.2 Globals and Functions (Pages)

When coding a form, the first statements define the global characteristics and includes. These are much like global variables and includes in C. The next thing defined is the first page of the form, which parallels a function in a C program. The table below demonstrates these similarities:

C	UFDL
`int version=3;`	`version="3.2.0";`
`char *bgColor="blue";`	`bgcolor=["blue"];`
`#include "header.h"`	`#include "header.frm"`

`void page_1()`
`<code>`
`}`
	`page_1=new page`
	`<code>`
	`}`

Notice the similarities in syntax as well. Statements are ended with a semi-colon. New pages (or functions) do not end with a semi-colon, and the code for each begins and ends with braces.

Within each page, UFDL allows the definition of page-specific characteristics similar to function specific variables. You can also define items, which can be thought of as instances of predefined structures or cases. Within each instance, there are a number of options that can be set. The following table illustrates this:

C	UFDL

`struct label{`
`char *value;`
`char *fontcolor;`
`char *bgcolor;`
`};`	

`void page_1()`
`{`
`struct label *label_1=NULL;`
	`page_1=new page`
	`{`

```
         char *bgcolor = "papayawhip";          bgcolor=["papayawhip"];

          label_1=(struct label*)malloc
```

```
      (sizeof(struct label));                    label 1_1=new label
      label_1->value=strdup("This is a label"); {
      label_1->fontcolor=strdup("white");  value="This is a label";
      label_1->transmit=strdup("none");    fontcolor=["white"];
      label_1->bgcolor=strdup(bgcolor);    transmit="none";
     }                                     }
```

In this example, value and fontcolor are options for the item label_1. Notice that they are contained within an opening and a closing brace, which makes it clear, without dereferencing, which item they belong to (this is similar to the with statement in Pascal). Also note that bgcolor is a page characteristic that is automatically assigned to label_1.

Additional pages can be defined just like additional functions, and there is no limit to the number of pages you can have, nor to the number of items you can have within a page. Additionally, it is possible to create custom items and options, allowing for greater flexibility.

C.3 References and Dynamic Option Reference

Specific items or option can be referenced through the use of a reference string. Additionally, the values of options can be dynamic option reference in a manner similar to C.

```
     C                     UFDL

Structure Declaration              Item declaration
struct label{                      label_1=new label
    struct label *value;           {
    char *fontcolor;                  value="label_2";
    };                                fontcolor=["white"];
                                   }
struct label *label_1;             label_2=new label
struct label *label_2;             {
                                      fontcolor=["blue"];
label_1->value=label_2;            }
label_1->fontcolor=strdup("white");
label_2->fontcolor=strdup("blue");

References                          References
label_1->fontcolor is white       label_1.fontcolor is white
label_1->value->fontcolor is blue label_1.value->fontcolor is blue

no equivalent                     page1.label_1.fontcolor is white
```

Note that the inclusion of page1 is optional. Including it allows the item or option to be referenced from a different page. Where C

requires you to pass values between functions by using parameters, UFDL allows you to access any value from any page on the form by using a direct reference.

C.4 Arrays

Many of the options in UFDL are set using array structures. Specific elements in these arrays are referenced just as they would be in C.

Array Declaration	Option Declaration
`char *format[2][2];`	`format=["integer",range=["1","10"]];`

```
format[0][0]=strdup("integer");
format[0][1]=NULL;
format[1][0]=strdup("1");
format[1][1]=strdup("10");
```

Element Reference	Element Reference
`format[1][0]`	`format[1][0]  or` `format[range][0]`

Note the second Element Reference in UFDL. This reference is based on assigning a name to an element or set of elements within the array. In this case, the name range was given to the second set of elements. This allows you to access the elements of an array without having to know the order of those elements.

There is no limit to the depth allowed an array in UFDL-you can have infinitely nested arrays. Additionally, UFDL does not create unnecessary blocks in memory. For instance, in the above example format[0][1] was assigned to be NULL in the C code. This is because this portion of the array is not used. In UFDL, that portion of the array is never created. This means that UFDL uses significantly less memory to store multi-dimensional arrays.

C.5 Assignment

UFDL has a single data type: the string. All values, be they integer, character, or float, are stored as literal strings. In order to assign literal strings, quotes are used just as in C. For example:

C	UFDL
`value=strdup("This is a literal string");`	`value="This is a literal string";`
`size[0]=strdup("10");` `size[1]=strdup("10");`	`size=["10","10"];`

Note that you do not need to use any of the string functions such as strdup. Memory management is handled automatically. Assigning a string to an option automatically copies the string and disposes of

any previous value for that option. Additionally, the full list of elements can be assigned to an array at any time in UFDL code. In C, this is only possible when you first declare your arrays (see

Absolute Positioning

Absolute positioning places items in set locations on the form. This form of positioning uses an x-y coordinate system to specify where, in relation to the upper-left corner of the form, the item should be placed. Absolute positioning allows for drag-and-drop form designer functionality. See Also: Relative Positioning

Automatic Action

Automatic actions are background actions that you can set your form to carry out without user prompting. These actions can be set to occur after a specific amount of time, or to repeat periodically. Use automatic actions to create effects such as periodic interfacing with databases. For example:

```
ping_action = new action
{
        delay = ["repeat", "120"];
        type = "submit";
        url = ["http://www.server.domain/cgi-bin/status"];
}
```

Alignment Modifiers

This is a group of modifiers used in relative positioning. They are used to set locators that align items in relation to each other. See Also: Relative Positioning, Locators

Bounding Box

An unseen rectangular area surrounding each item and including all elements of the item (including built-in labels and borders). Used in the relative ositioning scheme for determiniing the "edge" of an item.

Build Order

When you create items on a form, the order in which they are created forms a sequence. The first item you create is the first item in the sequence, the second item is second in the sequence, and so on. This sequence is called the build order. The build order can affect relative positioning.

Compression

You can set up forms to submit and save as compressed files. See the

descriptions or the saveformat and transmitformat options.

Computation

See: Formulas

Custom Item

Custom items are items that are not part of the standard UFDL. You define these items yourself. Custom items are ignored by the a UFDL parser, and are never visible on the form. Use custom items to integrate the form with other applications. See Also: Custom Options, Hidden Items

Custom Option

Custom options are options that are not part of the standard UFDL. You create these options yourself. Custom options are ignored by a UFDL parser, and ever affect the appearance of an item. Use custom options to integrate your form with other applications. See Also: Custom Items

Expansion Modifiers

This is a group of modifiers used in relative sizing. They are used to set locators that adjust the size of items in relation to other items on the form. See Also: Extent Modifier, Locators

Extent Modifier

This modifier is used in both normal and relative positioning. It is used to set locators that set the size of items in pixels. See Also: Expand Modifiers, Locators

Filtering

You can set up forms to filter out specific item types, option types, or items when they are transmitted. This can be useful for eliminating unnecessary information when the form is being transmitted to another application. See the descriptions for the transmit, transmititems, and ransmitoptions options. See Also: Compression

Formulas

Formulas allow you to add math or logic to your form. You can use them to add values or make decisions based on user input. Also referred to as computations or logical operations.

Global Settings

Global settings are used to set options for the whole form. Each item in the form will reflect that option setting, unless the same option is also set for that page or item.

Hidden Items

Hidden items are not independently visible on the form. However, in

some cases they may contribute to the appearance of other form items. The hidden items are: data items, cells, action items, help items, and custom items.

Identifiers

Identifiers are used to uniquely identify a page, item, option, or option element in the form. For a discussion of the character set you may use when creating identifiers, see the section called Identifiers in '2. The Universal Forms Description Language'.

Include

The #include statement allows you to "include" external files in a form. These files must be in a UFDL format that is compatible with the form. A UFDL parser must insert include files into forms at the place marked by an #include statement.

Input Focus

When a user views a form, the input focus is the focus that moves from item to item in the form when the user presses the TAB key.

Item Tag

Each item has a unique item tag, or name, which is used to identify that item.

Locators

Locators are used to set both the locations of an item on the page and any relative or extent sizing that should apply to the item.

Modifier

Modifiers are combined with item tags to create locators. Modifiers describe how an item should be positioned in relation to its reference items. For example, after, below, left-to-left, and so on. See Also: Locators, Reference Items

Page

Each form can be composed of any number of pages, just like a paper form. Each page is identified with a page tag, begins with an open brace ({) and ends with a close brace (}). All forms must have at least one page.

Page Settings

These settings set global option settings for all items on the page (unless an individual item description overrides the setting). Page settings override global settings, but are overridden by options set

for individual items.

Page Tag

Each page has a unique page tag, or name, which is used to identify that page.

Reference

References allow you to identify a specific option by providing a "path" to it. This means that you can refer directly to a specific option anywhere in the form. A reference is constructed by combining the page tag, item tag, and option name that will point to the option you want. For example, page1.title_label.value points to the value option of the title_label on page one.

Reference Items

When using relative positioning or sizing, you will need to use some items as reference points. These reference points will be used as anchors that set either where an item is positioned on the form or how large the item is. If the reference items are moved or change size, the item using them as anchors may also move or change size. See Also: Relative Positioning, Relative Sizing

Relative Positioning

Relative positioning places items on the form in relation to other items. This means that items will move on the form if their reference item moves. Relative positioning is useful for ensuring cross-platform compatability. See Also: Absolute Positioning, Reference Items

Relative Sizing

Relative sizing adjusts the size of items on the form in relation to other items. This means that items will change size on the form if their reference item moves or changes size. Relative sizing is useful for ensuring that the edges of your items line up on the form. See Also: Reference Items

Required Status

The required status is part of the format option. It determines whether the user of a form is required to enter information into an item. The required status can be set to "optional" or "mandatory".

Start Value

The start value is an element of the option name that contains the literal resolution of an option reference or other formula. The start value element is represented as an open angle bracket, the value in quotation marks, and a close angle bracket on the left-hand side of the equal sign, like this:

```
value<"Jane E. Smith"> = page1.nameField.value;
```

The viewer sets this literal value when a form is signed, submitted,

or saved (and discards any old value if necessary). Because a digitally signed formula never fires after being signed, the start value for the option is always the same--and therefore it is possible to reference the option and get the signed literal value.

Tab Order

This is the order in which the user will move through the item on the form by pressing the TAB key. The tab order only includes those items that take user input, such as fields, buttons, and so on. You can set the tab order yourself.

Version Number

This option records the version number of the UFDL that was used to create the form. It is only available in the global settings.

Author Contact Information

David Manning
dmanning@uwi.com

voice. 250-479-8334
fax. 250-479-3772
post. David Manning
UWI.Com
1095 McKenzie Avenue, 4th Floor
Victoria, B.C., Canada
V8P 2L5

Expires: February 04, 1999