

**Oracle[®] Retail Design
Configuration Guide
Release 12.0
May 2006**

Copyright © 2006, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

| | |
|---|------------|
| Preface | vii |
| Audience | vii |
| Related Documents | vii |
| Customer Support | vii |
| 1 Introduction | 1 |
| General User View Management | 2 |
| Tab Layout Definition..... | 3 |
| Spectrum Utility..... | 3 |
| Product Information Administration..... | 3 |
| Server Side Reporting Template Administration | 4 |
| 2 General User View Configuration | 7 |
| Style File Tab Definition | 8 |
| Style File Tab Configurable Attributes | 8 |
| Configuration of the Oracle Retail Design to Oracle Retail Webtrack Project Integration | 10 |
| Functional Description of the “Projects” Parameter Group | 10 |
| Project Interface Configuration | 12 |
| Parameters | 13 |
| Enabling of the Comments Entry Capability | 17 |
| Comments Entry Definition Files | 17 |
| Comments Entry Configurable Attributes | 20 |
| Server Side Reporting Definition..... | 21 |
| Scope Definition | 23 |
| 3 Tab Layout Configuration | 25 |
| Spectrum Utility..... | 25 |
| Product Information Administration | 25 |
| Getting Started | 26 |
| Field Elements | 27 |
| Sample XML File | 28 |
| Elements and Attributes | 30 |
| Identifying Statement..... | 30 |
| SpecSheets | 30 |
| SpecSheet..... | 31 |
| Page | 34 |
| Matrix | 35 |
| cellattrs..... | 36 |
| Column | 37 |
| ColumnSet | 37 |
| Heading..... | 38 |
| RowSet..... | 39 |
| Cell | 39 |
| CellChoice | 40 |
| CalcSet..... | 41 |
| Form | 41 |
| itemattrs | 43 |
| Defaults..... | 45 |
| Common Initial Value | 45 |
| Simple Item Types – TextField and IntField | 45 |
| FloatField..... | 45 |
| TextArea | 46 |
| Icon | 46 |
| Checkbox | 47 |

| | |
|---|-----------|
| Label | 47 |
| MultiLabel | 47 |
| DateField | 48 |
| Custom | 48 |
| Parameter | 48 |
| SubForm | 49 |
| Choice | 50 |
| Option | 50 |
| Image | 51 |
| Calc | 51 |
| 4 XFO Templates | 53 |
| XFO Introduction | 54 |
| XFO Operation | 54 |
| Basic Structure | 54 |
| Expressions and Attributes | 54 |
| Example: | 54 |
| SF Processing Elements | 55 |
| sf:str | 55 |
| sf:int | 55 |
| sf:float | 55 |
| sf:date | 55 |
| sf:set | 56 |
| sf:update | 56 |
| sf:func | 57 |
| sf:if | 57 |
| sf:for | 57 |
| sf:macro | 58 |
| sf:call | 58 |
| Builtin Values and Functions | 58 |
| array(n) | 58 |
| geticon(string) | 59 |
| getprop(prop) or getprop(prop, deflt) | 59 |
| valuekey(v) | 59 |
| hasmorevalues(set) | 59 |
| XFO and Styles | 60 |
| Values | 60 |
| Functions | 66 |
| Standard Properties | 69 |
| Configuring XFO Objective Sheet Output | 69 |
| Configuring XFO Client Printing in Oracle Retail Design | 69 |

| | |
|--|-----------|
| 5 Spreadsheet Expression Syntax | 71 |
| Data Types | 71 |
| Lists..... | 72 |
| Arrays | 72 |
| Array items | 72 |
| Object Values | 73 |
| Variable Names | 73 |
| Function Calls..... | 73 |
| Expressions | 74 |
| Built-in Functions | 77 |
| Style Linkages..... | 84 |
| Style Custom Spec Sheet Fields..... | 87 |
| Retailer Status | 87 |
| Supplier Status | 87 |
| Colour List | 87 |
| Size Range | 88 |
| Documents | 88 |
| Images..... | 89 |
| Product Type and ELC Type | 89 |
| Comment Display | 89 |
| A Appendix: designconfig.dtd | 91 |
| B Appendix: specsheets.dtd | 97 |

The purpose of this Oracle Retail Design Configuration Guide is to provide a reference for enterprise administrators who develop XML and XFO-based forms.

Audience

Anyone with an interest in developing a deeper understanding of the configuration capabilities surrounding Oracle Retail Design will find valuable information in this guide.

Related Documents

If you wish to find further information, see the following applicable Oracle Retail documents:

- Oracle Retail Design Online Help
- Oracle Retail Design User Guide
- Oracle Retail Design Operations Guide
- Oracle Retail Design Release Notes
- Oracle WebTrack Release Notes
- Oracle WebTrack Online Help
- Oracle WebTrack User Guide
- Oracle WebTrack Configuration Guide
- Oracle Retail Retail Server Installation Guide
- Oracle Retail Retail Server Data Model

Customer Support

- <https://metalink.oracle.com>

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step-by-step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Introduction

Oracle Retail Design is a collaborative product development solution that provides retailers considerable flexibility in order to meet their business requirements. In addition to the configuration available via the standard options with the Design Administration Console, the administrator can define a specific format or definition for use within the application in the following three areas:

- General User View Management
- Tab Layout Definition
- Server Side Reporting Templates

Each of these areas requires a definition to be created and uploaded within Oracle Retail Design. The general user view management is controlled by uploading an XML-based file via the Configurations option within the Oracle Retail Design administration console. Oracle Retail Design administrators leverage the Spectrum utility or Product Information administration to upload XML-based files to support the tab layout definition. Finally, the server side reporting templates can be uploaded via a specific URL accessed from the Oracle Retail Design administration console. Server side reporting templates can be defined in an XFO-based file.

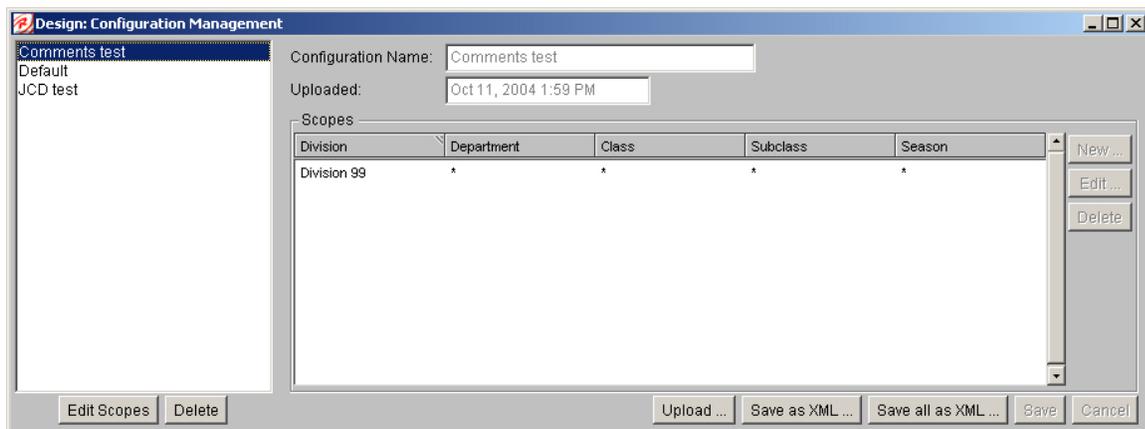
The purpose of this document is to provide a reference for enterprise administrators who develop the XML and XFO-based forms. Oracle Retail assumes that they will have access to the following resources:

- Administrator access to Oracle Retail Design and Spectrum
- The designconfig.dtd file, which defines the data types used in developing the XML-based forms for the general user view configuration.
- The specsheet.dtd file, which defines the data types used in developing the XML-based forms for tab layout definition.
- Oracle Retail Design User Guide
- Oracle Retail Design Operations Guide

Oracle Retail also assumes that client administrators have a high-level understanding of XML and XFO file structures.

General User View Management

The Configurations option on the Oracle Retail Design administration console is used to control features of the Oracle Retail Design user view. The primary purpose of the configuration management is to control which tabs are present for a particular scope. In addition, it is used to set various configurable enterprise parameters including the Oracle Retail Webtrack project integration rules, comments entry, and which server side printing format files are available in the client printing function within Oracle Retail Design. In order for the user to define the configuration, the administrator needs to upload the .xml-based form within Oracle Retail Design. Once the configuration is uploaded, the administrator has the flexibility to further define the scope that the configuration should support. Once the configuration files have been uploaded, there are supporting format files and specification files that may need to be uploaded.



Configuration Management Window in Oracle Retail Design

The actual development of the XML files can be performed in any text editor. A text editor that supports XML would be most efficient and is recommended. The primary functions supported within the XML files are highlighted in the chapter “General User View Configuration.”

The elements, attributes, and syntax requirements of this file have been defined in the designconfig.dtd available in “Appendix A – designconfig.dtd” and by browsing to the following URL:

- <http://www.retail.com/import/dtds/designconfig.dtd>.

Tab Layout Definition

Oracle Retail Design is a collaborative product development solution that provides retailers flexibility to capture key product information by product type. Specifically, retailers have the ability to configure the Specification, Bid, and Estimated Landed Cost (ELC) tabs within Oracle Retail Design according to the different types of products that they will be developing with Oracle Retail Design. In addition, they have the ability to configure new tabs to appear within a style file in Oracle Retail Design.

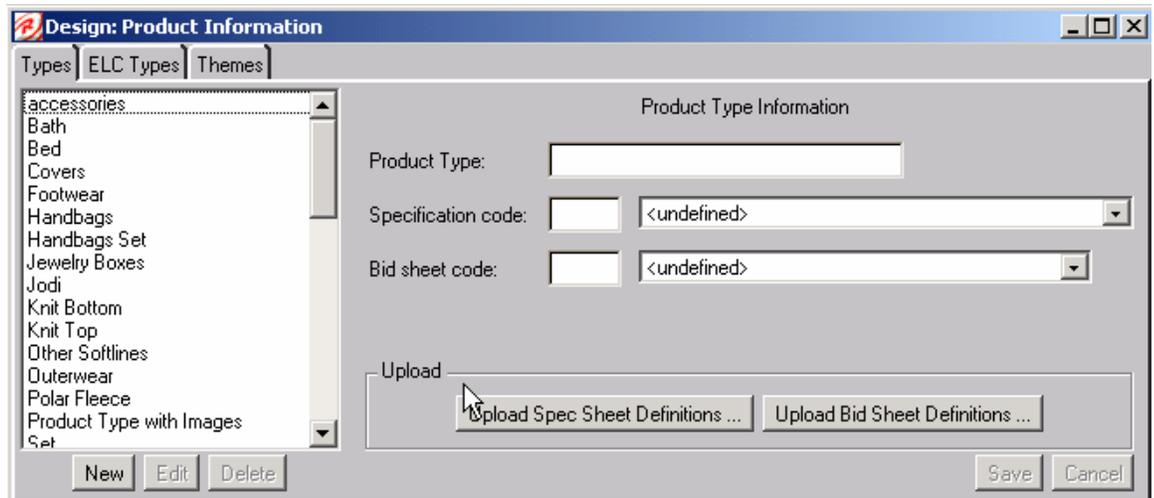
In order to design the user interface and define the field and validation rules used within a configured tab, XML-based forms need to be created, uploaded into Oracle Retail Design, and assigned to specific product types defined by the retailer's administrator. The XML tag language works with the application logic to generate and display specified forms within the configurable tabs. In addition to displaying a configurable form with data fields, images, and icons, the application allows data fields to be linked, calculated, and exported within the configurable tabs.

Spectrum Utility

The Spectrum utility is used to support the development and display of the configurable forms. It is a separate application that appears only for Oracle Retail Design administrators dedicated to supporting the development and display of configurable tab layout sheets. This service is organized to allow the administrator to upload and view specification, bid, estimated landed cost and general tabs during development. In addition, the administrator has the ability to manage the icons that may be configured to appear within a specific configurable form. Specific details of how to navigate through Spectrum are included in the Oracle Retail Design User Guide.

Product Information Administration

The Product Information option on the Oracle Retail Design administration console is where the configurable XML files are uploaded and linked to a specific product or ELC type within Oracle Retail Design. In order for the user to view and test all functionality included with the configurable forms, the user needs to upload the form within Oracle Retail Design and assign it to the appropriate types via Product Administration. Once XML files are uploaded and linked to a specific type within Oracle Retail Design, the user is able to see the type and select it within the product file. The specific product type that is selected within a product file in Oracle Retail Design determines what is populated within the Specification and Bid tabs. Similarly, the specific ELC type that is selected within Oracle Retail Design determines what appears in the ELC tab. The Product Information option is only designed to support the specification, bid and ELC configurations. Other general tab definitions need to be managed within Spectrum.



Product Information Window in Oracle Retail Design

The actual development of the XML files can be done in any text editor. A text editor that supports XML would be most efficient and is recommended. The tags and attributes that are used within the development of the XML files are defined in the specsheet.dtd and have been annotated in the chapter “Tab Layout Configuration.” A copy of the specsheet.dtd is available in “Appendix B – specsheet.dtd” and by browsing to the following url:

- <http://www.retail.com/import/dtds/specsheet.dtd>

Server Side Reporting Template Administration

Oracle Retail Design supports the generation of printable output on the client and server side. The client side printing is generated based on specified rules within the code. If users are expecting an exact format, they may not choose to leverage the client side print. The server side printing leverages specific format files that have been defined and uploaded within Oracle Retail Design. The server side printing is available from within the client side printing option dialog box and is also used during the technical specification export process supported by Oracle Retail Integrator. Format files can be developed using XFO technology and can be uploaded by browsing to the following URL from within the Oracle Retail Design Administration Console:

- <https://www.retail.com/applications/design/template.jsp>

The administrator is prompted to browse and upload the template file, define the format that it uses and identify a mode that will be used to cross-reference the configuration file. The template mode is a free-form text field and the value input is used to cross-reference the uploaded file in other configuration administration steps. Specifically, the mode is used within the general user view configuration file to identify a format that can be used by the server side printing options available to the user. In addition, the mode can be referenced within Oracle Retail Integrator as the ObjectiveSheetType during the setup of the run type used to support the technical specification export process.

Note: Although the objective sheet template upload process continues to support the upload of .xmf files, Oracle Retail recommends that xfo formats be used.

Objective Sheet template upload

File:

Template type:

Template mode:

Browse to required file, and press **Upload** to transmit file to server.

Enter a template *mode* when uploading templates for special situations, including style file reporting. The mode should consist of letters, digits and period characters.

Print Template Upload Window in Oracle Retail Design

The actual development of the XFO files can be done in any text editor. A text editor that supports XML would be most efficient and is recommended. The primary functions supported within the development of the XFO files are highlighted in chapter “XFO Templates.”

General User View Configuration

The Configurations option on the Oracle Retail Design administration console is used to control features of the Oracle Retail Design user view. The configuration file has the following primary purposes:

- **Style file tab definition**
Within the configuration file, the administrator can identify the tabs that will appear within a style file. These tabs can be a mixture of standard tabs and new-user defined tabs.
- **Configuration of the Oracle Retail Design to Oracle Retail WebTrack project integration**
There are a number of aspects of the project integration that can be configured within the configuration file. For example, the administrator has the ability to create projects by style or by style/color, identify specific field mapping that should occur during project integration, and identify fields that should be allowed to be updated by the integration. These options are all supported within the configuration definition file.
- **Enabling of the comments entry capability**
Because the comments dialog box that appears within Oracle Retail Design is configurable, there are setup steps required to enable the standard comments functionality within Oracle Retail Design. The configuration file includes the reference to the comments dialog layout definition file and includes the layout used to support the printing of comments. Once the configuration file is updated and uploaded, the administrator uploads the comments dialog layout definition within Spectrum.
- **Server side reporting definition**
As mentioned in the previous chapter, Oracle Retail Design supports the generation of printable output on the client using pre-defined format files. The server side reporting option is available from within the client side printing option dialog box and leverages specific format files that have been defined and uploaded within Oracle Retail Design. During the upload of the format files, a mode is assigned to that format. To enable this user to select this format within the printing option dialog box, the mode needs to be assigned to a user-friendly option name within the general user view configuration.

Each of these features can be defined at an enterprise level, or the configuration file can define these options for a particular scope, based on the season or department/division of a style. This ability allows the administrator to define the features differently for different departments or divisions, or support changes to the features for a new season. Oracle Retail recommends that enterprise-specific features be defined early in the form followed by scope-specific settings, allowing you to define all user view configuration options within one definition file. The ability to upload multiple definition files and set the scope outside of the definition file is also supported.

This chapter highlights the functional areas supported by user view configuration file and discusses how to enable these functions using the XML definition file. The elements, attributes, and syntax requirements of the XML definition file have been defined in “Appendix A – designconfig.dtd” and by browsing to the following URL:

- <http://www.retail.com/import/dtds/designconfig.dtd>

Style File Tab Definition

There are two required definition files that must be updated to support the configurable style file tab definition. First, the 'user view' configuration file needs to be updated to identify the tabs that should appear within a style file. Once this definition is completed, it must be uploaded via the Oracle Retail Design Administration – Configurations option. See the Oracle Retail Design User Guide for information about how to upload a file within the Configurations option. Once the configuration file is updated, the administrator must upload the supporting tab layout definition referenced within the configuration file via Spectrum. Specifically, the Oracle Retail Design administrator opens the “Design general tab sheets” application within the Spectrum and uploads the .xml file that defines the new tab layout. For more information on defining the tab layouts, please see the chapter “Tab Layout Configuration.” Once all of these steps have been completed, the user should be able to see the new tabs within Oracle Retail Design.

Style File Tab Configurable Attributes

Within the configuration file, the administrator can identify the tabs that should appear within a style file. In addition to the standard Summary screen, the following three types of tabs can be configured within the definition file to appear in the style files:

- One type of tab represents the standard tabs (other than the Summary screen) that already exist within Oracle Retail Design.
- The two other types of tabs represent new tabs that could be defined and set up to appear within a style file.

The resulting configuration could be a combination of the standard Summary tab, new tabs, and a subset of standard tabs that already exist within a style file today.

A reference to a builtin tab corresponds to the standard tabs other than the Summary screen that exist within a standard style file. The existence of a builtin tab indicates that the referenced standard tab should be present within the new configuration of the style file. Examples of builtin tabs include: volumes, price, bom, cost, specification, bid, etc, and labels. A specific builtin tab cannot be referenced more than once within a configuration. Because the configuration has priority over the view security settings within Oracle Retail Design, a standard tab must be included within the configuration to appear within the style file. The user permissions of each tab will still be supported by the Oracle Retail Design administration – Security option.

The new tabs that can be configured to appear are defined within the configuration file as either spec or custom tabs. Spec tabs represent tabs that can have their layout configured by the administrator and uploaded within Spectrum. There are additional attributes of the spec tab including user permissions, the name that should appear on the tab, and the sheet type number that corresponds to the XML definition uploaded within Spectrum. Custom tabs reference tabs that are defined by a Java class name without configurable definition files. In the case of the custom tabs, they are supported by custom-written Java code and cannot easily be changed. The majority of administrators leverage the spec tabs when creating user-defined tabs to appear within a style file. Custom tabs are not supported on a www.retail.com deployment of Oracle Retail Design.

Within the configuration, if the standard Summary screen is used, it must be the first tab that appears within the list. The administrator does have the ability to configure a new summary screen using the new user-defined spec tab option.

Below is an example excerpt of a configuration file that leverages the standard Summary screen and references two new user-defined spec tabs:

```
<Configuration name="Default+Comments+Calendar">
  <Details>
    <Tabs>
      <Summary/>
      <Builtin type="volumes"/>
      <Builtin type="elc"/>
      <Builtin type="specification"/>
      <Builtin type="bid"/>
      <Builtin type="labels"/>
      <Spec key="comms" type="301">
        <TabLabel>Comments</TabLabel>
      </Spec>
      <Spec key="timeandaction" type="1000">
        <TabLabel>Calendar</TabLabel>
      </Spec>
    </Tabs>
  </Details>
</Configuration>
```

In this example, the Comments and Calendar tabs are configured to appear within the style file. To complete the setup and ensure the layout exists, the administrator uploads the corresponding XML files with sheet types of 301 and 1000 via the Design general tab sheets option within Spectrum.

Below is an example excerpt of a configuration file that replaces the standard Summary screen within a configuration summary screen and supports the remaining standard tabs:

```
<Configuration name="Configurable Summary Screen">
  <Details>

    <Tabs>
      <Spec key="configsumm" type="1">
        <TabLabel>Summary</TabLabel>
      </Spec>
      <Builtin type="volumes"/>
      <Builtin type="price"/>
      <Builtin type="bom"/>
      <Builtin type="cost"/>
      <Builtin type="elc"/>
      <Builtin type="specification"/>
      <Builtin type="bid"/>
      <Builtin type="labels"/>

    </Tabs>

  </Details>

</Configuration>
```

To complete the setup and ensure that the layout exists, the administrator uploads the corresponding XML files with a sheet type of 1 via the Design general tab sheets option within Spectrum.

Additional parameters can be used to support the user-defined tabs. For example, specific user permissions using the perm attribute could be defined to identify which users have the ability to edit the tab.

In addition, the Conditions feature could be used to identify which fields need to be entered before the Save button is enabled. This feature is especially useful to support the configurable summary screen tab because it allows you to identify any required fields on the style file.

As explained earlier, the set of tabs can be defined for a particular scope. For more information, see the section, ‘Scope Definition’ included within this chapter.

Configuration of the Oracle Retail Design to Oracle Retail Webtrack Project Integration

There are a number of aspects of the project integration that can be configured within Oracle Retail Design including the ability to specify field mapping that should occur, the ability to create projects by style or by style/color, and the ability to identify fields that should be allowed to be updated by the integration. These options are all supported within the configuration definition file supported by the designconfig.dtd.

Functional Description of the “Projects” Parameter Group

Integration Mapping

Oracle Retail Design and Oracle Retail WebTrack contain built-in mappings of Design style elements to WebTrack project information. These mapping rules can be changed by definitions within the “projects” parameter group in a configuration. As with all parameter groups, it may appear within a single configuration, or be placed at the end of the file and shared amongst several configurations using “ref” and “id” attributes.

Default Configuration

The elements below are currently assigned as defaults from Oracle Retail Design to Oracle Retail WebTrack. If the configuration does not include a “projects” group, these default assignments remain unchanged. If there is a “projects” group, the mappings it contains replace the default assignments.

| Design Element | Corresponding WebTrack Element |
|-------------------------|---|
| Short Name | Project Name (required for project creation) |
| Department | Department |
| Order Required By Date | Completion Date (required for project creation) |
| Style/Color | Project Number (required for project creation) |
| Sell price * Quantity | Value |
| Unique Id | Style Id |
| Project Created by User | Comment |

Email Notification

If the XML produces any errors during the WebTrack project creation or update, an applicable email is sent to the administrator.

Ongoing Updates

The design configuration file provides an option to allow subsequent changes to individual data elements within Oracle Retail Design to impact/update their mapped counterparts in Oracle Retail Webtrack.

If a data element within Oracle Retail Design is configured to allow updates to the mapped data elements within Oracle Retail WebTrack, all projects, tracks and events associated with that unique style/color are automatically updated accordingly. For example, if the Short Name is changed in Oracle Retail Design, the project name reflects those changes in the Oracle Retail WebTrack project and tracks records associated to that project.

When a style is edited within Oracle Retail Design, all elements that are eligible for update from Oracle Retail Design to Oracle Retail WebTrack are updated in Oracle Retail WebTrack accordingly. Specifically, if there are five elements flagged for update from Oracle Retail Design to Oracle Retail WebTrack and only one of those elements is actually changed, all five elements are updated. Any manual changes made (to those elements within Oracle Retail WebTrack that can be modified) are overridden by the subsequent update within Oracle Retail Design.

Updates are made to all style/color combinations within Oracle Retail WebTrack for the style that is being updated in Oracle Retail Design. If the style is configured to group all colors under one project, that project alone is affected by the updates made within Oracle Retail Design. If a color is added to Oracle Retail Design for a style that is associated with existing Oracle Retail WebTrack projects at the style/color level, an applicable style/color project is also added to Oracle Retail WebTrack. Other style/color combinations for the style are not affected.

If the style has been copied or retrieved in Oracle Retail Design, resulting in new Oracle Retail WebTrack entries with the same project name and/or design ID, only the new Oracle Retail WebTrack project created from the updated (that is, copied or retrieved) style is updated in Oracle Retail WebTrack.

'Required by Dates'

If the 'Required by Date' is changed in Oracle Retail Design, and this element is configured to update the final event within a track for the style, all previous planned dates for uncompleted events are updated based on the lead times.

The final event date in Oracle Retail WebTrack is *only* changed if the final event date in Oracle Retail WebTrack is the same as the 'Required by Date' in Oracle Retail Design.

Delete Functionality

The administrator has the ability to set a business rule that states whether or not a style or color that is deleted in Oracle Retail Design results in corresponding projects and tracks' being deleted in Oracle Retail WebTrack. In order for a delete to occur in Oracle Retail WebTrack, the style is required to be in an applicable workflow status. If no configuration is provided for this option, the default setting is 'off,' and deletes do not take place in Oracle Retail WebTrack.

Note that a project is not re-created in Oracle Retail WebTrack in the following scenario:

- When a style-level Oracle Retail WebTrack project that was created from Oracle Retail Design is deleted manually using the Oracle Retail WebTrack program, and a change is then made to the style in Oracle Retail Design.

Multiple and Split Tracks

If a Oracle Retail WebTrack project maps to multiple tracks, all tracks for that project are updated accordingly.

In the event that a track is split and subsequently the project name is changed through Oracle Retail Design, only that portion of the split track name that matches the original name is changed to the new name.

For example, if a track name is 'Cat Sweaters,' and you split the track, selecting Forest Green for the color, the new track name is 'Cat Sweaters/Forest Green (1).' Changing the project name in Oracle Retail Design to 'Cat Warmers' results in the following:

- The original track is renamed to 'Cat Warmers.'
- The split track is renamed 'Cat Warmers/Forest Green (1).'

Diary Entries

Diary entries are made for any changes made to a style in Oracle Retail Design that result in changes made to a track name within Oracle Retail WebTrack.

Project Interface Configuration

This configuration is accomplished by adding a "projects" ParameterGroup to the Details section of a Oracle Retail Design configuration. If you want different project setups for different scopes, add a separate projects section to each configuration. Alternately, you can have a common projects group at the end and refer to it with the ref/id attributes, as in the following:

```
<Details>
  <!-- Tab, etc details -->
  <ParameterGroup ref="commonprojects"/>
</Details>
</Configuration>
<!-- Shared parameters -->
<ParameterGroup id="commonprojects" name="projects">
  ... project configuration ...
</ParameterGroup>
```

If a projects parameter group is absent from the configuration, the current built-in mappings and rules are used.

If a projects parameter group is present, none of the built-in mappings are used. All the required mappings and update rules must be specified.

Parameters

The project interface recognizes the following parameters in the projects parameter group:

projectlevel: This parameter controls whether projects are created at the style or color level. The value of the parameter should be 'style', 'colour' or 'color'. If omitted, projects are created at color level, as in the original implementation.

For example:

```
<Parameter name="projectlevel">style</Parameter>
```

deleteprojects: This parameter controls whether projects and tracks are deleted when colors are removed from a style (if the project level is color) or when the style is deleted. The value should be 'true', 'false', 'yes' or 'no'. The default is false; projects are not deleted.

For example:

```
<Parameter name="deleteprojects">yes</Parameter>
```

mappings: This parameter is compulsory because it is where all the mappings and update rules are defined. The parameter's value is a multi-line Java properties file. The following is a complete example:

```
<Parameter name="mappings">

# Functions

func.vatify    = $1 * 1.175

# Basic project data

projectname    = style$shortname
projectnumber  = style$stylenumber || '/' || colourname
duedate        = style$orderby
value          = vafify(style$price)
comments       = 'Created from style file by ' || user
information    = style$agent

# Validity check

valid = isset(style$supplier) & amp; isset(style$theme)

# Some attributes

attr.1.name     = Theme
attr.1.value    = style$theme
attr.1.required = true

attr.2.name     = ELC target
attr.2.value    = style$elctarget
attr.2.type     = f

# Extras

extra.1         = ifset(style$supplier, 'No supplier!')
extra.2         = style$buyer

# Name and number can be updated

update.projectname = true
update.projectnumber = true
update.duedate     = true
```

```

update.extras      = true

</Parameter>

```

The following types of information are defined by the properties:

1. Functions
Property names starting with func. are used to define functions which can be used in expressions. The example above shows a simple function which multiplies by 1.175 to apply Value Added Tax (VAT) at the standard rate for the United Kingdom.
2. Basic mappings
These basic mappings define the values for basic project data. The property names are the following:

| Property name | Meaning | Required? |
|---------------|----------------------------|-----------|
| projectname | Project name | Yes |
| projectnumber | Project number | Yes |
| duedate | Project due date | Yes |
| value | Project value | No |
| comments | Project comments | No |
| information | 'Order' information string | No |
| valid | Validity flag | no |

The value of each property is an expression in the standard syntax. It can refer to the **style\$** link values which are available in spec sheets, along with these special names:

| Name | Meaning |
|----------------|--------------------------------------|
| user | Name of current user |
| userenterprise | Name of current user's enterprise |
| useremail | Current user's email address |
| true or yes | 1 |
| false or no | 0 |
| colourname | Color name |
| colourcode | Color code |
| colourvolume | Quantity for color from volume sheet |

Note: Please see the Style Linkages section within the chapter "Spreadsheet Expression Syntax."

The three color-related names are not available if projects are created at style level.

Mappings for the first three items are required; the configuration is rejected if any are absent. If the expression evaluates to undefined or empty values, no project creation or updating takes place.

The special name ‘valid’ can be used as an additional control on whether projects should be created or updated. If the expression evaluates to undefined or zero, no project creation takes place.

3. Attributes

Attributes are arbitrary items attached to the project (strictly to the project’s pseudo-order item) which can be displayed by the Details button on a track.

The project interface always creates two standard attributes listing the style ID and supplier; using *attr.* properties, additional attributes can be added to the project.

Attribute properties start with *attr.N* where N are consecutive integers starting at one. The interface scans for *attr.1.name*, *attr.2.name*, and so on, stopping when a property is not found.

Each attribute is defined by a *name*, which is the display string shown in the GUI, and a *value*, which is an expression using the same rules as for other mappings. As shown below, an attribute can have a *format* attribute which defines the type of the value and how it is displayed:

| Format string | Meaning |
|---------------|--|
| t | Text (this is the default) |
| i | Integer |
| fN | Floating point value displayed with N decimal places |
| d | Date |

If the format is not t, the value is treated as a number.

4. Extra values

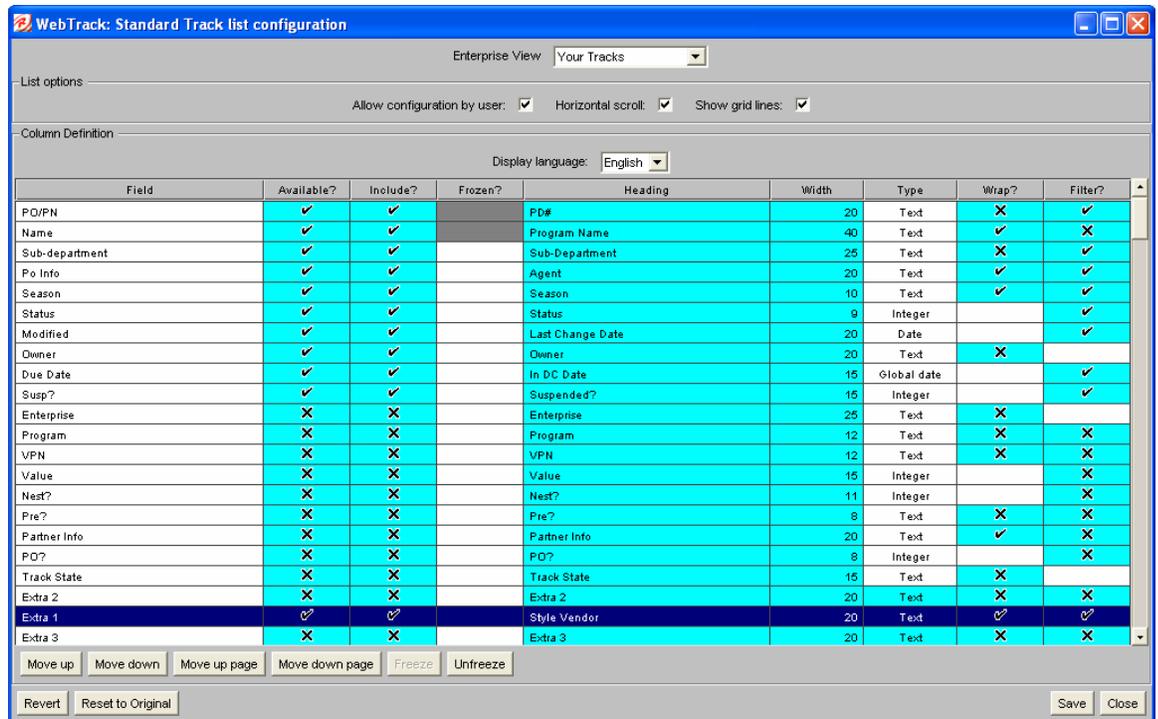
Project ‘extra’ values are numbered values which are stored in a specific area of the database. The track list can be configured to display these values in the same way as ‘misc’ values are handled in the Oracle Retail Design style list. Currently, the tracks list handles up to ten extras.

An ‘extra’ value is defined by a property named *extra.N* where N is the extra number (1-10).

Here is an example mapping the style supplier to a track list field called ‘Style Vendor,’ then displaying it in the tracks list screen. In the ‘projects’ parameter group, the following line would create the mapping:

```
# Extras
  extra.1      = style$supplier
```

Then the webtrak administrator would configure the Webtrak ‘Lists’ as below, mapping *extra.1* to a field called ‘Style Vendor.’



WebTrack Standard Track List Configuration

5. Update flags

‘Update’ properties are flags that control whether project fields are updated when a style is edited. The value of the property is an expression; update is selected if the expression evaluates to a defined non-zero value. Allowing the flags to be defined by expressions means that updates can be controlled on a style-by-style level if required. All update flags are unset by default.

The property `update.all` can be used to set the flags for all mappings; individual `update.x` properties can then act as an override.

There is an update flag for all the basic mappings (projectname, and so on) and also flags for attributes (`update.attrs`) and for extras (`update.extras`). If attribute or extra updating is selected, all attributes or extras are updated on an edit.

Enabling of the Comments Entry Capability

The ability to enter comments within Oracle Retail Design provides an ongoing dialog and enables further collaboration between partners on a specific style file. Because the comments dialog box that appears within Oracle Retail Design is configurable, there are setup steps required to enable the standard comments functionality within Oracle Retail Design. This section identifies the components required to set up the functionality and the attributes and functions that support the comments configuration.

Comments Entry Definition Files

There are two required definition files that need to be configured to set up the comments entry capability. First, the user view configuration file is updated to identify the layout of the comments dialog and the printing of comments. Once this definition is completed, it needs to be uploaded via the Oracle Retail Design Administration – Configurations option. See the Oracle Retail Design User Guide for information on how to upload a file within the Configurations option. Once the configuration file is updated, the administrator uploads the comments dialog layout definition referenced within the configuration file via Spectrum. Specifically, the Oracle Retail Design administrator must open the “Design general tab sheets” application within the Spectrum and upload the .xml file that defines the comments dialog layout. To ensure that the comments button is visible within Oracle Retail Design, the final step requires the administrator to verify the security settings for the comments functionality via Oracle Retail Design administration – Security. Once these steps have been completed, the user should have the ability to see the comments functionality within Oracle Retail Design.

Note: There is also a default comments.xfo file that is already uploaded for enterprises accessing www.retail.com. This file defines the server side format file that supports the comments printing. All that is required to leverage this file is to reference the comments mode within the user view configuration file. See the examples below.

Below are examples of default configuration files available to support the standard setup of the comments entry capability. The files can be also be found by browsing to the following URL:

- <http://www.retail.com/import/dtds/>

This example configuration file references the comments layout and printing capability. You will note that the commentspectype which references the comments dialog layout file type is 300. For the comments entry capability to work, the administrator must upload an xml file with a spec sheet type of 300.

```
<?xml version="1.0"?>
<!DOCTYPE Configurations PUBLIC "-//Oracle retail.com//DTD Design configuration
XML//EN" "http://www.retail.com/import/dtds/desconfig.dtd">

<!-- Default configuration: allow parameters to be added -->

<Configurations>

  <!-- The global default -->

  <Configuration name="Default">
```

```
<Details>

  <!-- Styles and comments reporting parameters -->

  <ParameterGroup ref="reporting" />

  <!-- Comments definition -->

  <ParameterGroup ref="comments" />

</Details>

</Configuration>

<!-- Shared reporting parameters -->

<ParameterGroup name="reporting" id="reporting">

  <Parameter name="type">xfo</Parameter>

  <ParameterGroup name="styles">

    <Parameter name="layout">landscape</Parameter>

  </ParameterGroup>

  <ParameterGroup name="comments">

    <Parameter name="modes">
Text View/comments,prop.format=text
List View/comments,prop.format=list
    </Parameter>

    <Parameter name="layout">portrait</Parameter>

  </ParameterGroup>

</ParameterGroup>

<!-- Shared comments parameters -->

<ParameterGroup name="comments" id="comments">
  <Parameter name="commentstype">300</Parameter>
</ParameterGroup>

</Configurations>
```

Here is an example of the comments dialog layout must be uploaded within the “Design general tab sheets” of Spectrum. This file is named commentsentry.xml. Please note that the sheet type is 300. These types of files are described in more detail in the chapter “Tab Layout Configuration.”

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SpecSheets PUBLIC "-//Oracle retail.com//DTD Design specsheets XML//EN"
"http://www.retail.com/import/dtds/specsheet.dtd">
<SpecSheets application="styletab">

  <!-- Configurable comments entry: standard version -->

  <SpecSheet type="300" description="Commnts entry: version1" fill="true" version="1"
borders="etched">

    <Page scrollable="false">

      <Form name="all">

        <!-- Set confirm option for multiple styles -->

        <Calc cell="comm$confirm" expr="comm$multi"/>

        <!-- Set autoemail option to send email always -->

        <Calc cell="comm$autoemail" expr="1"/>

        <SubForm fill="b">
          <Form name="outer" visible="comm$canadd" description="New comment" columns="2">

            <TextField label="Subject" fill="h" limit="32" cell="comm$subject"/>
            <Checkbox label="Send as email" cell="comm$email"/>
            <Label fontsize="-5" cellw="2" fill="n"/>

            <!-- Main text -->

            <TextArea width="155" rows="15" fill="b" cell="comm$text" cellw="2"/>

          </Form>
        </SubForm>

        <!-- Display -->

        <SubForm fill="b">

          <Form name="display" visible="comm$multi = 0" description="Current comments">
            <Custom name="comments" fill="b">
              <Parameter name="mode">both</Parameter>
              <Parameter name="initial">text</Parameter>
              <Parameter name="textfontsize">2</Parameter>
              <Parameter name="textsize">900x275</Parameter>
              <Parameter name="listrows">15</Parameter>
            </Custom>
          </Form>

        </SubForm>

        <!-- Magic 'mark as unread stuff' -->

        <Form name="marker">
          <Checkbox label="Mark comments as unread" pos="w" fill="h" cell="comm$unread"/>
        </Form>

      </Form>

    </Page>

  </SpecSheet>

</SpecSheets>
```

Comments Entry Configurable Attributes

To support the configuration of `commentsentry.xml`, there are a number of attributes that specifically support the comments capability. These are also referenced within the chapter “Spreadsheet Expression Syntax.”

Within the `commentsentry.xml` definition, the standard `style$` link fields (see the Style Linkages section in the chapter “Spreadsheet Expression Syntax”) are available if comments for a single style are being entered. In addition, the following comment fields are supported:

Output fields:

comm\$subject: set this to the ‘subject’ for the comment. The value can come from a text field or from a standard set in a drop down.

comm\$text: set this to the ‘text’ for the comment. This will usually be set from a TextArea field but could be calculated from other values.

comm\$email: set this to non-zero if the comment is also to be sent as an email. The standard email entry dialogue will be shown to collect the addresses. This will normally be set from a check box.

Input fields:

comm\$multi: this is set to 1 if comments for multiple styles are being entered. It can be used to control the visibility of the comments display area. This will always be empty for multiple styles.

comm\$canadd: this is set to 1 if the user is allowed to create comments. It should be used to control display of the entry fields. There is no point in showing these if the user cannot add.

The `comments` component is used to display style file comments. It can be used as part of the comment entry dialog form to display the current set of comments. There is no input value associated with the component, so a cell name should not be present.

```
<Custom name="comments">
  <Parameter name="mode">text, list or both</Parameter>
  <Parameter name="initial">text or list</Parameter>
  <Parameter name="print">>true or false</Parameter>
  <Parameter name="textfontsize">N</Parameter>
  <Parameter name="textsize">WxH</Parameter>
  <Parameter name="listrows">N</Parameter>
  <Parameter name="showent">other, partner, always or never</Parameter>
</Custom>
```

| Parameter | Meaning | Default |
|-----------|---|---------|
| mode | If <i>text</i> , the comments are shown in text format, most recent first; if <i>list</i> , the comments are displayed in multi-column sortable list format. If <i>both</i> , radio buttons are available to switch between the formats. | both |
| initial | If the mode is <i>both</i> , this selects the initial display | text |
| print | Selects whether the separate Print Comments button is available. (Note that if server-side comment reporting is not configured, comments are always printed in text format and the mode must be <i>text</i> or <i>both</i> to enable printing). | true |

| Parameter | Meaning | Default |
|--------------|--|---------|
| textfontsize | A positive or negative increment which is added to the point size of the default font to get the font for text format displays. For example if the value is 2, the font will be 2 points larger than the default. | 0 |
| textsize | The dimension in pixels of the text display area. This may be adjusted to change the size of the comments entry dialogue. The actual size may be affected by screen size or by the size of the list format component. | 900x275 |
| listrows | The number of visible rows in the list format display (note that this refers to rows with a single line of text). The actual number of rows displayed may be larger if the comments component is stretched to fit a form or if the text format is made larger. | 15 |
| showent | <p>Controls whether the enterprise of the comment user is shown in brackets after the user name:</p> <p><i>other</i>: the enterprise is shown if it is not the same as that of the current user.</p> <p><i>partner</i>: the enterprise is shown if the user belongs to a trading partner.</p> <p><i>always</i>: the enterprise is always shown.</p> <p><i>never</i>: the enterprise is never shown.</p> <p>The value of this parameter is passed to the server template as the showent property.</p> | other |

Server Side Reporting Definition

Oracle Retail Design supports the generation of printable output on the client using pre-defined format files. The server side reporting option is available from within the client side printing option dialog box and leverages specific format files that have been defined and uploaded within Oracle Retail Design. During the upload of the format files, a mode is assigned to that format. To enable this user to select this format within the printing option dialog box, the mode needs to be assigned to a user-friendly option name within the user view configuration. This section identifies the configuration attributes required to set up the modes within the user view configuration file. The actual definition of the format files is covered in the chapter “XFO Templates.”

To configure XFO printing for styles, a *reporting* **ParameterGroup** should be included in your configuration file. If the setup is common to all of your configurations, it can then be included as a shared group at the end of the file and use a 'ref' attribute to refer to it:

```
<ParameterGroup ref="reporting">
...
...

<!-- Shared comments and styles reporting parameters -->

<ParameterGroup name="reporting" id="reporting">

  <Parameter name="type">xfo</Parameter>

  <ParameterGroup name="styles">

    <Parameter name="modes">
Mode one/simple
Mode two/two
  </Parameter>

  </ParameterGroup>

  <ParameterGroup name="comments">

    <Parameter name="modes">
Text View/comments,prop.format=text
List View/comments,prop.format=list
  </Parameter>

    <Parameter name="layout">portrait</Parameter>

  </ParameterGroup>

</ParameterGroup>
```

The reporting parameter group contains the following two nested groups:

- styles, which sets up styles printing from the main Print button
- comments, which sets up comments printing.

The type parameter specifies the reporting type; this should be xmf or xfo. XFO is the Oracle Retail recommended print format. If specified outside the inner groups, it applies to both groups.

The optional modes parameter defines the templates modes (as in xfo-mode-19). The modes are listed on separate lines. The string before the / is shown in the drop down in the print dialogue box. The string after the / is the mode string assigned when uploading the format file. In the example above, this configuration causes two options to appear within the drop-down list when the server side reporting option is selected within the client side Print dialog. Specifically, Mode one and Mode two are available for selection. The content of Mode one will be tied to the format file that was uploaded with mode set to "simple", while the content of Mode two is tied to the format file uploaded with mode set to "two." As the administrator uploads new printing format files, he or she must update the configuration file for them to appear to the user.

The mode strings may be followed by optional parameters which are passed to the formatting engine. In the example here, the same template is used for both modes; tests within the template control whether the output is in text or list format.

There is a built-in option `max` which specifies the maximum number of style files which are supported by the mode. This can be used to prevent hugely-detailed reports being generated for large numbers of styles. For example:

```
<Parameter name="modes">
  Mode one/simple
  Mode two/two,max=1
</Parameter>
```

Here the second mode is available only when a single style file is selected.

If the layout parameter is present, the Portrait/Landscape option is included in the print dialogue; the value of the parameter specifies the default for the option.

Scope Definition

The configuration file can be defined at an enterprise level, or the configuration file can define the options described above for a particular scope. This choice allows the administrator to define the features differently for different departments or divisions, or support changes to the features for a new season. Oracle Retail recommends that both enterprise-specific features are defined in the form followed by scope-specific settings. This strategy allows you to define all user view configuration options within one definition file. The ability to upload multiple definition files and set the scope outside of the definition file is also supported. This section identifies the configuration attributes required to set the scope within the user view configuration file.

Below is an example excerpt of a configuration file that identifies how the scope can be set within the configuration file for a specific division and season.

```
<Configurations>

  <!-- The default for a Config Summary Screen for Accessories and Spring 2006 -->

  <Configuration name="Configurable Summary Screen">
    <Scopes>
      <Scope>
        <Division>
          <Name>Functional Accessories</Name>
        </Division>
        <Season>
          <Name>SP06</Name>
        </Season>
      </Scope>
    </Scopes>

    <Details>

      <Tabs>

        <!-- Standard tabs -->

        <Spec key="configsumm" type="1">
          <TabLabel>Summary</TabLabel>
        </Spec>
```

```
<Builtin type="volumes"/>
<Builtin type="price"/>
<Builtin type="bom"/>
<Builtin type="cost"/>

<Builtin type="elc">
  <Parameter name="autocreate">>false</Parameter>
</Builtin>

<Builtin type="specification"/>
<Builtin type="bid"/>
<Builtin type="labels"/>

</Tabs>
```

If a configuration file is uploaded to replace the previous configuration, and it does not contain the same scope set as the previous version, the old scope configurations are lost. If a configuration with a new name is uploaded, existing configurations in the same scopes are not removed. Oracle Retail Design uses the most recent upload for a specific scope. Where there are multiple configurations and scopes within Oracle Retail Design, the configuration is determined based on the following rules:

1. Find best hierarchy match.
2. If there is more than match, choose configuration with matching season.
3. If there is still more than one match, choose most recently uploaded version.

The hierarchy match is based on the selections on the user console. If the enterprise is using subscoping at the subclass level, but the user does not select a subclass, the match is by division, department and class only. Note that Oracle Retail Design deployed on www.retail.com does not use subscoping, and therefore the scope cannot be matched at the class or sub-class level.

Tab Layout Configuration

Oracle Retail Design is a collaborative product development solution that provides retailers flexibility to capture key product information by product type. Specifically, the retailer has the ability to configure the Specification, Bid, and Estimated Landed Cost tabs within Oracle Retail Design according to the different types of products they will be developing with Oracle Retail Design. In addition, they have the ability to configure new tabs to appear within a style file in Oracle Retail Design.

In order to design the user interface and define the field and validation rules used within a configured tab, XML-based forms need to be created, uploaded into Oracle Retail Design, and assigned to specific product types defined by the retailer's administrator. The XML tag language works with the application logic to generate and display specified forms within the configurable tabs. In addition to displaying a configurable form with data fields, images, and icons, the application allows data fields to be linked, calculated, and exported within the configurable tabs.

This chapter provides information about the following:

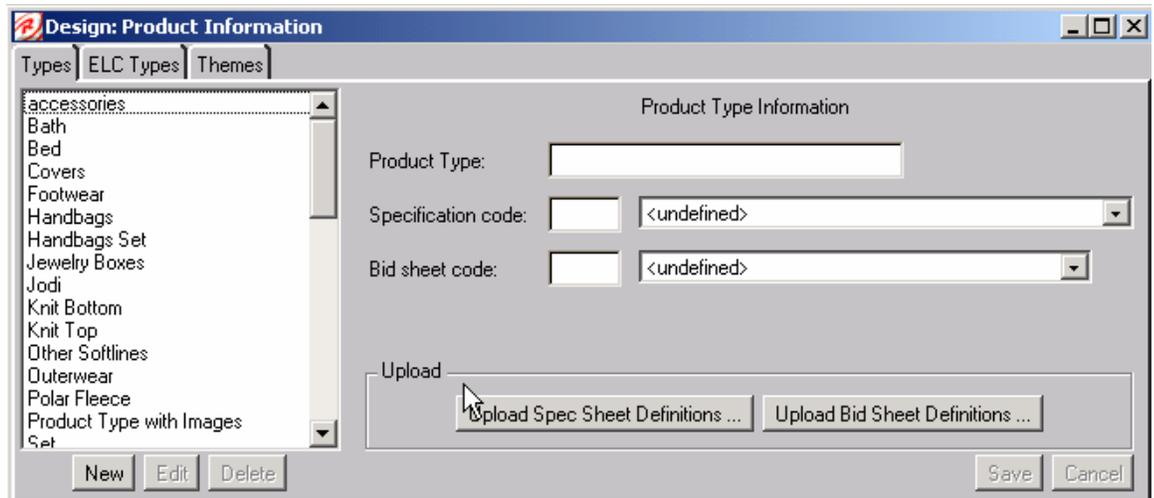
- A sample configuration file.
- How to upload the file into Oracle Retail Design.
- Detailed explanations of attributes and elements used to support the configuration.

Spectrum Utility

The Spectrum utility is used to support the development and display of the configurable forms. It is a separate application that appears only for Oracle Retail Design administrators. This service is organized to allow the administrator to upload and view specification, bid, estimated landed cost and general tabs during development. In addition, the administrator has the ability to manage the icons that may be configured to appear within a specific configurable form. Specific details of how to navigate through Spectrum are included in the Oracle Retail Design User Guide.

Product Information Administration

The Product Information option on the Oracle Retail Design administration console is where the configurable XML files are uploaded and linked to a specific product or ELC type within Oracle Retail Design. In order for the user to view and test all functionality included with the configurable forms, the user needs to upload the form within Oracle Retail Design and assign it to the appropriate types via Product Administration. Once XML files are uploaded and linked to a specific type within Oracle Retail Design, the user is able to see the type and select it within the product file. The specific product type that is selected within a product file in Oracle Retail Design determines what is populated within the Specification and Bid tabs. Similarly, the specific ELC type that is selected within Oracle Retail Design determines what appears in the ELC tab.



Product Information Window in Oracle Retail Design

The actual development of the XML files can be done in any text editor. The tags and attributes that are used within the development of the XML files are defined in the designspecsheets.dtd. The elements and attributes of this file have been annotated below in the Elements and Attributes section.

Getting Started

Depending on the sheet that will be supported, the administrator can either start development of a specification sheet from scratch or copy an existing sheet if the information will be similar. As the sheet is being developed, the administrator should upload the sheet into Spectrum to verify the code will be compiled without error. If an error is encountered and the administrator has been uploading the sheet fairly frequently, the administrator should be able to pinpoint the section of code that will need to be reviewed and changed.

As you are editing your XML document, keep in mind that all XML tags need a beginning and ending tag. This is especially important to note if you are developing the xml sheet from scratch.

Before you actually begin your new specification sheet, you may want to go through the exercise of laying out the various pages, forms and subforms to help organize your overall approach to developing the sheets. Once you define the headings and appropriate layout, the following questions should be asked when completing section of data:

- What type of data field? (Date, Integer, Text, Text Area, Choice, FloatField, and so on)
- What is the name of the data field?
- What is the width of the data field?
- If no name exists, will the field need to be recorded in the log file. What name would you like the field to be recorded under in the log file?
- Does this field need to be linked to another standard field or specification field within Design?
- Does this field need to appear on the list screen within Design?
- Does this field need to appear in the season details file as a mapped field?
- Who has the ability to edit/update this field?

- Is this a calculated field? Will this field be used in calculations?

Field Elements

Within the configurable specification sheet language, the retailer has the ability to configure fields to appear within the list screen, link configured fields between sheets, link configured fields to the Summary window and identify configured fields to be exported within the season details file or as part of the server-side print file. To achieve these functions, the administrator must use various types of fields including standard fields, miscellaneous fields, and mapped fields.

- **Standard fields** – These are standard database fields that can be referenced when linking information from the summary screen or linking information from a configurable tab to the summary screen. Because standard fields are already defined in the database, in many cases, they are already available to appear in the list screen configuration and are available to be exported. See the sections concerning style linkages in the chapter “Spreadsheet Expression Syntax” for details related to Standard Fields.
- **Miscellaneous fields** – These are configured fields that can be referenced to perform client-side functions. Specifically, if an administrator would like a configured field to appear on the style file list screen, it is necessary for the administrator to tie the configured field to a miscellaneous field. In addition, to link configurable data between configurable tabs, miscellaneous fields must be used. Currently, Oracle Retail Design also includes miscellaneous fields on the season details extract and the server-side print, but in the long term, miscellaneous fields will be used as an exclusively client-based configurable field. There is a limit of 250 miscellaneous fields that can be configured within the Lists Administration to appear within the style file list screen. There is no limit to the number of miscellaneous fields that are used to link data between tabs. However, the same miscellaneous field cannot be used more than once within the same form.
- **Mapped fields** – These are configured fields that can be referenced to perform server side functions. Specifically, if an administrator would like a configured field to appear on the season details extract and/or to be included in the server side print, mapped fields are recommended. There is no limit to the number of mapped fields that are referenced within the configurable forms. However, the same mapped field cannot be used more than once within the same form.
- **Linked Fields** – In Oracle Retail Design, a function within the XML-based configurable forms language links data between the specification, bid, and estimated landed cost configurable tabs. This “link” function behaves very similarly to the linking provided by the existing miscellaneous fields; however there is no limit to the amount of data that can be referenced by the link functionality. In addition, data referenced by the link functionality can be used in calculations and expressions. Data referenced by the link functionality alone cannot be configured to appear on the list screen or the server side reporting functions.

Sample XML File

The following excerpt provides an example of the XML code for a simple Oracle Retail Design specification sheet. It incorporates many of the most common XML tags, or elements and attributes, which are used to define a spec sheet tab. You can use this sample file and the descriptions of the elements and attributes in this chapter to explore the opportunities available for developing your own configurable tabs.

```
<?xml version = '1.0'?>
<!DOCTYPE SpecSheets PUBLIC "-//retail.com//DTD Design specsheets XML//EN"
"http://dem19.retail.com/retailservera/import/dtlds/specsheet.dtd">
<SpecSheets application="styles">

  <SpecSheet type="501" description="Apparel Specification" tabbed="true"
fill="true">
    <Page title="Summary">
      <Form name="Summary">
        <Form name="STmain" columns="2">
          <Form name="STpanel1" columns="2" description="Header Summary">
            <TextField label="Product ID" width="20"
expr="style$stylenumber"/>
            <TextField label="Class" width="20" expr="style$class"/>
            <TextField label="Description" width="20"
expr="style$shortname"/>
            <TextField label="Sub Class" width="20" expr="style$subclass"/>
            <TextField label="Department" width="20"
expr="style$department"/>
            <TextField label="Product Type" width="20" expr="style$type"/>
            <TextField label="Season" width="20" expr="style$season"/>
            <TextField label="Size Range" width="20"
expr="style$sizerange"/>
            <TextField label="Collection" width="20" expr="style$theme"/>
            <TextField label="Base Size" width="20" cell="style$misc_111"/>
            <Custom label="Status" width="20" expr="style$cust_stateid"
name="rstatus"/>
            <DateField label="Approval" mode="local"/>
          </Form>
          <Form name="STlogo">
            <Icon iconname="demologo"/>
          </Form>
          <Form name="STpanel101" columns="1">
            <Image dim="300x300" cellh="50">
              <ImageOption name="image1"/>
            </Image>
          </Form>

          <Form name="STpanel101a" description="Product Details" columns="2">
            <TextField label="Block Reference" width="15"
cell="style$dispatch"/>
            <TextField label="Testing Standard" width="15"
cell="style$misc_102"/>
            <Choice label="Product Life" width="20" cell="style$misc_105">
              <Option value="">&lt;Select</Option>
              <Option>Carryover</Option>
              <Option>New</Option>
              <Option>Exit</Option>
            </Choice>
            <Choice label="Silhouette" width="20" cell="style$misc_100">
              <Option value="">&lt;Select</Option>
              <Option>Darted</Option>
              <Option>Pleated</Option>
              <Option>Flare</Option>
            </Choice>
          </Form>
        </Form>
      </Form>
    </Page>
  </SpecSheet>
</SpecSheets>
```

```

        <Option>Cropped</Option>
        <Option>A Line</Option>
        <Option>Empire</Option>
        <Option>Halter</Option>
        <Option>Sheath</Option>
        <Option>Slip</Option>
        <Option>Strapless</Option>
        <Option>Wrap</Option>
    </Choice>
    <Choice label="Fit" width="20" cell="style$misc_101">
        <Option value="">&lt;Select></Option>
        <Option>Classic</Option>
        <Option>Low Rise</Option>
        <Option>Slim</Option>
        <Option>Stretch</Option>
        <Option>Loose</Option>
    </Choice>
    <Choice label="Fabric" width="20" cell="style$category">
        <Option value="">&lt;Select></Option>
        <Option>Angora</Option>
        <Option>Boucle</Option>
        <Option>Cashmere</Option>
        <Option>Cotton</Option>
        <Option>Cotton Sateen</Option>
        <Option>Cotton Twill</Option>
        <Option>Cotton Poplin</Option>
        <Option>Crepe</Option>
        <Option>Linen</Option>
        <Option>Silk Chiffon</Option>
        <Option>Silk Dupioni</Option>
    </Choice>
    <TextField cell="style$misc_9" label="Fiber" map="fibertgt"
width="15"/>
    <TextField label="Weight" width="15" cell="style$misc_10"/>
    <TextField label="Construction" width="15"
cell="style$misc_11"/>
    <TextField label="Yarn Count" width="15" cell="style$misc_12"/>
    <TextField label="Gauge" width="15" cell="style$misc_13"/>

    <Choice label="Finish" width="20" cell="style$misc_14">
        <Option value="">&lt;Select></Option>
        <Option>Bio Polish</Option>
        <Option>Mercerized</Option>
        <Option>Micro Sanded</Option>
        <Option>Soft Handfeel</Option>
        <Option>Wrinkle-Resistant</Option>
    </Choice>
    <Choice label="Wash" width="20" cell="style$misc_68">
        <Option value="">&lt;Select></Option>
        <Option>Compact Finish</Option>
        <Option>Fabric Wash</Option>
        <Option>Mild Silicone</Option>
        <Option>Non-Wash</Option>
        <Option>Colorfast</Option>
        <Option>Soft Handfeel</Option>
    </Choice>
    <Choice cell="style$misc_114" keyed="true" label="Fabric Country
Origin Target" map="fabriccotgt" perm="2,9,12" width="20">
        <Option key="y" list="fabricco"></Option>
    </Choice>
    <Choice cell="style$misc_174" keyed="true" label="Dye
House/Finisher Target" map="fabricmilltgt" perm="2,9,12" width="20">
        <Option key="y" list="dyehouse"></Option>

```

```

        </Choice>
    </Form>
</Form>

```

Elements and Attributes

This section provides a reference for elements and attributes used in defining Oracle Retail Design configurable specification sheets or tab layouts. Elements are presented in the order in which they appear in the specsheet.dtd file. For each element, a description, format, and where applicable, an example is provided.

Identifying Statement

Every specification sheet will start with a statement in this format:

```

<?xml version="1.0" encoding="UTF-8"?>

<!-- DTD for Design spec sheets.  Version 0.01 -->

```

Example

This identifies the DTD file from which the specification sheet will be validated.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SpecSheets PUBLIC "-//www.retail.com//DTD Design specsheets XML//EN"
"http://www.retail.com/import/dtds/specsheet.dtd">

```

SpecSheets

SpecSheets is the enclosing element. It contains one or more individual spec sheet definitions.

```

<!ELEMENT SpecSheets (SpecSheet+)>
<!ATTLIST SpecSheets application CDATA #IMPLIED
                version      (1 | 2) "2">

```

The application attribute defines the internal “application code” for the sheets. The sheet processor can reject an import if it is configured for a different code. There are three options for the Specsheets application value:

- styles – This maps to the specification sheet. If this is uploaded as a bid tab or ELC tab, an error will occur.
- stylebids – This maps to the specification sheet. If this is uploaded as a spec tab or ELC tab, an error will occur.
- styleelc – This maps to the specification sheet. If this is uploaded as a spec tab or bid tab, an error will occur.
- styletab – Maps to a ‘general’ sheet which can be configured as a new tab, comments display, and so on.

The version attribute can define a default version for all the sheets in the file. The default is 2. If you were to change the version to 1, the information would display in a different format (see below). Note that the border and background attributes can be used to override the default settings.

Example

```
<SpecSheets application="styles" version="1">
<SpecSheets application="styles" version="2"> OR <SpecSheets
application="styles">
```

Results of the different SpecSheets version attributes:

SpecSheets Version 1 Format

SpecSheets Version 2 (Default) Format

SpecSheet

SpecSheet defines a single sheet layout. It contains one or more pages.

```
<!ELEMENT SpecSheet (Page+)>
```

```
<!ATTLIST SpecSheet type          CDATA          #REQUIRED
                    description    CDATA          #REQUIRED
                    tabbed         (true | false)  "false"
                    defaulttabs   (true | false) "false"
                    fill          (true | false)  "false"
                    version       (1 | 2)        #IMPLIED
                    borders       (none | lined | etched) #IMPLIED
                    background    CDATA          #IMPLIED
                    enabled       CDATA          #IMPLIED
                    perm          CDATA          #IMPLIED>
```

The integer **type** attribute defines the specification type number.

The **description** attribute defines text displayed for the sheet in administration tools.

If the **tabbed** attribute is present, tabs are displayed; if **defaulttabs** is present the default tab titles are used. This attribute is deprecated and will be removed eventually.

The **fill** attribute defines whether the sheet fills horizontally. The default is false.

The **version** attribute defines the spec sheet display version. Currently versions 1 and 2 are supported. The default is the SpecSheets version attribute.

The **borders** attribute sets the default border for all forms in the sheet; if omitted the default is 'etched' for version 1 and 'lined' for version 2.

The **background** attribute sets the background color for the page and its forms. It should be a color name like 'black' (see the fields of the `java.awt.Color` class), or an RGB value in decimal or hex (decoded by `java.lang.Integer.decode`). If the **background** attribute is not present, the background color is 'light gray' (the standard Design color) for version 1 and white for version 2 sheets.

The **enabled** attribute defines an expression which can be used to dynamically update the 'editing enabled' state of the sheet. The attribute is available on most nested elements; the most recently seen value will override the settings on enclosing elements. Note that enabled settings do not combine. A form may have a 'true' enabled attribute while the value for the sheet is false; the form will then be available for editing.

The **perm** attribute defines editing permissions for the specsheets. The basic format of the perm value is:

allow-deny

'allow' and 'deny' are comma-separated lists of application-specific security roles. The roles are the keys defined in the system administration services window (the 'sec:' prefix may be omitted). We use the following table in defining permissions:

| Service Administration – User Types Set-Up | | |
|--|-----------------------|-------------------|
| Key | Description | Enable by Default |
| sec:1 | Account Manager | |
| sec:2 | Administrator | |
| sec:3 | Artist | |
| sec:4 | Imports | |
| sec:5 | Buyer | |
| sec:6 | IS | |
| sec:7 | MIO | |
| sec:8 | Planning/Allocation | |
| sec:9 | Product Management | |
| sec:10 | Quality Control | |
| sec:11 | Oracle Retail Support | |
| sec:12 | Sourcing Manager | |
| sec:13 | Technical Designer | |
| sec:14 | Packaging | |
| sec:loc | [All local users] | X |

| Service Administration – Partner Roles Set-Up | | |
|---|---------------------|-------------------|
| Key | Description | Enable by Default |
| sec:p1 | Agent | |
| sec:p2 | Supplier | |
| sec:p5 | Domestic Supplier | |
| sec:prt | [All partner users] | X |

“*” means all users; it should not be used in conjunction with any other roles. An empty list means no users.

“deny” may be omitted if there are no exclusions.

Examples of perm attributes:

1. perm=“*” means ‘everybody’ and has the same effect as omitting the attribute.
2. perm=“loc-1” means all users in the ‘loc’ group except those in the local ‘1’ group. It is equivalent to perm=“sec:loc-sec:1”.
3. perm=“*-2,p1” means everybody, except those users in the local ‘2’ group and the partner ‘1’ group.

Common settings will be perm=“loc” for local users only and perm=“prt” for partner users only.

Not all combinations are sensible, for example, “loc-*” means ‘all local users except for everybody’, and is the same as perm=“”. An item that cannot be edited by anyone is not very useful.

The **perm** attribute is available on most other nested elements; the most recently seen value will override settings on enclosing elements.

Example

```
<SpecSheet type="20" description=" Shirt Specification Sheet" tabbed="true"
perm="2,9">
```

In this example, the type appears within Spectrum and the administration console and is the unique identifier or key that defines the sheet. If a specification sheet with the same type is uploaded, the contents of the current type will be replaced.

The description appears within Spectrum and the administration console, but is not a key for the sheet. Many sheets can have the same description.

Oracle Retail recommends using the tabbed attribute instead of the default tabs attribute. If you change a specification sheet to read tabbed=“false”, it assumes there are not multiple tabs and only the last tab would appear when uploading the sheet into Spectrum.

The permissions attribute defined here allows the Administrator (2) and Product Management (9) to have access to information within this SpecSheet.

Page

The Page element defines a single page or tab.

```
<!ELEMENT Page ((Matrix | Form)*)>
<!ATTLIST Page title CDATA #IMPLIED
               scrollable (true | false) "true"
               background CDATA #IMPLIED
               enabled CDATA #IMPLIED
               perm CDATA #IMPLIED>
```

If the sheet is not tabbed, there must be one page only. Each Page contains one or more matrices or forms.

The **title** attribute defines the tab title. It is required if the sheet is tabbed.

The **scrollable** attribute defines whether the entire page scrolls. The default is true.

The **background** attribute sets the background color of the page; if omitted, the default for the sheet is used.

The **enabled** attribute defines an expression which can be used to dynamically update the 'editing enabled' state of the sheet. The attribute is available on most nested elements; the most recently seen value overrides the settings on enclosing elements. Note that enabled settings do not combine. A form may have a 'true' enabled attribute whilst the value for the sheet is false; the form will then be available for editing.

Example

```
<Page title="Construction Details">
</Page>
<Page title="Bill of Materials ">
</Page>
<Page title="Graded Measurements">
</Page>
```

In this example, there are three pages within this SpecSheet titled Construction Details, Bill of Materials, and Graded Measurements. If tabbed<>true, only one page would appear. As mentioned above, it would be the Graded Measurements page that would appear. It is very important to include the end tag for each page, as there are multiple forms, matrices defined within a page.

Because you may not always know how much space the contents of the new specification sheet may take, set the scrollable attribute to true (the default).

Matrix

The Matrix element defines a multi-column list.

```
<!ELEMENT Matrix ((Calc | CalcSet)*, CellChoice?, (Column | ColumnSet)+, (Row | RowSet)*)>
<!ATTLIST Matrix name                CDATA                #REQUIRED
                  description         CDATA                #IMPLIED
                  border              (none | lined | etched) #IMPLIED
                  rows                CDATA                #REQUIRED
                  headings            (false | true)       "true"
                  visiblerows         CDATA                #IMPLIED
                  visiblecols         CDATA                #IMPLIED
                  scrollable           (false | true)       #IMPLIED
                  horizontalscroll    (false | true)       "false"
                  leftfixedcols       CDATA                #IMPLIED
                  rowheadingwidth     CDATA                #IMPLIED
                  showeditable        CDATA                #IMPLIED
                  visible              CDATA                #IMPLIED
                  cellpfx             CDATA                #IMPLIED
                  map                  CDATA                #IMPLIED>
```

The **name** attribute is the unique name for this matrix in the sheet. The name is always required. The name is not defined as an XML ID because the same name may be used in different SpecSheets.

The **description** attribute is used to generate a heading for the matrix.

The **rows** attribute defines the number of rows; the number of columns is implied by the number of Column elements.

The **headings** attribute defines whether headings are displayed.

The **visiblerows** attribute defines the number of rows displayed in the client GUI. The default is 5. The actual number of rows displayed may differ if the matrix size is altered to fit in the available space.

If the **scrollable** attribute is absent, the matrix scrolls vertically if the Page is not scrollable, and does not scroll if the Page is scrollable. This default behavior is suitable for fixed matrices, but may not be best for matrices with dynamic row sets.

The **horizontalscroll** attribute specifies whether a horizontal scroll bar is present in the Matrix. This can be useful if there is a dynamic column set. If horizontal scrolling is selected, the **visiblecols** attribute gives an indication as to the visible width of the matrix. The measure is based on a notional default column width and does not relate directly to real columns in the matrix.

The **leftfixedcols** is the number of non-scrolling columns on the left of the matrix. It can be used only if **horizontalscroll** is true and must be less than the number of columns. A ColumnSet counts as a single column in the **leftfixedcols** count.

The **rowheadingwidth** attribute must be present if row headings are required; it is the width of the headings column.

If **showeditable** is true, editable cells are displayed with a different background color in edit mode.

The **visible** attribute defines an expression which can be used to dynamically show or hide the matrix. The matrix is visible if the expression evaluates to a non-zero number.

The **cellpfx** attribute must be set if any cells in the matrix are involved in spreadsheet calculations. It is used as a prefix to form the spreadsheet cellnames for matrix cells. A cell at row 'r' and column 'c' will have a spreadsheet cellname of pfx\$r.c. For example, if the prefix is "zz", the cell at row 3, column 6 will have a spreadsheet cellname of zz\$3.5. If no cells are involved in calculations, omitting the cellpfx attribute speeds up several matrix operations. The **map** attribute defines a mapping for the entire matrix.

Initial contents and row/cell mappings are defined by **Row** elements.

Calc elements may be included before **Column** elements to define intermediate values and functions.

The optional **CellChoice** element defines the default items for any choice columns or cells in the matrix. It can be included even if the default cell type is not chosen; this allows the default to be used for a number of individual choice cells without repeating the items.

CalcSet elements are used to compute 1-d and 2-d array values in dynamic row and/or column areas.

Example

This example also uses the Form element. See later elements for more information about its usage.

```
<Form name="BOMmain" description="Bill of Materials Details">
  <Matrix name="BOMpanell" rows="5">
    <Column width="15" heading="FABRIC"/>
    <Column width="15" heading="Where is it Used?"/>
    <Column width="15" heading=" Content"/>
    <Column width="15" heading="Supplier Name"/>
    <Column width="15" heading="Cost"/>
    <Column width="15" heading="Weight"/>
    <Column width="15" heading="Construction"/>
  </Matrix>
</Form>
```

cellattrs

The *cellattrs* entity is a shorthand for repeated attribute definitions in the DTD. These attributes can be set in Matrix, Column, or Cell elements. The Cell setting overrides the Column setting, which in turn overrides the Matrix setting.

```
<!ENTITY % cellattrs 'align (l | c | r) #IMPLIED
                    type (text | int | float | date | checkbox | choice) #IMPLIED
                    prec CDATA #IMPLIED
                    limit CDATA #IMPLIED
                    enabled CDATA #IMPLIED
                    perm CDATA #IMPLIED'>
```

```
<!ATTLIST Matrix %cellattrs;>
```

The **align** attribute defines the column alignment. For example, 'l' would be used for text items, 'r' for numeric values and 'c' for date values. The alignment can be overridden by an align attribute in a Cell element.

The **type** attribute defines the datatype expected for values in the column.

The **prec** attribute can be used with float only; it defines the number of decimal places; the default is 2.

The **limit** attribute can be used with text cells only; it defines the maximum text length. It must be 'none' (to cancel an enclosing default limit) or a positive integer.

Column

The column attribute defines a single column. The width is required, but the heading is optional. If the heading is omitted but the matrix headings attribute is true, an empty heading is displayed.

```
<!ELEMENT Column (Heading?, CellChoice?)>
<!ATTLIST Column width      CDATA          #REQUIRED
                  heading   CDATA          #IMPLIED
                  headingexpr (false | true) "false"
                  map        CDATA          #IMPLIED
                  %cellattrs;>
```

The column heading may be set in a **heading** attribute or child Heading element, but not both.

If the **headingexpr** attribute is true, the heading value is treated as an expression which is evaluated to obtain the column heading.

The **map** attribute defines a column level mapping.

The optional **CellChoice** element defines the default items for choice cells in the column.

Example

```
<Column width="10" heading="POM"/>
<Column width="20" heading="Description"/>
<Column width="10" heading="Tot(-)"/>
<Column width="10" heading="Tot(+)" />
<Column width="10" heading="XS"/>
<Column width="10" heading="S"/>
<Column width="10" heading="M"/>
<Column width="10" heading="L"/>
<Column width="10" heading="XL"/>
```

ColumnSet

The ColumnSet attribute defines a dynamic ‘column’ set. This contains a set attribute defining an expression which should evaluate to a list of column headings. A ColumnSet is treated as a single ‘virtual’ column when referred to in spreadsheet expressions and Cell elements. The spreadsheet value associated with a cell in a dynamic column set will be a 1-d or 2-d array.

```
<!ELEMENT ColumnSet (CellChoice?, SubColumn*)>
<!ATTLIST ColumnSet width      CDATA          #IMPLIED
                    set        CDATA          #REQUIRED
                    dimension  CDATA          #IMPLIED
                    required   (false | true) "false"
                    map        CDATA          #IMPLIED
                    %cellattrs;>

<!ELEMENT SubColumn (CellChoice?)>
<!ATTLIST SubColumn width      CDATA          #IMPLIED
                    heading   CDATA          #REQUIRED
                    map        CDATA          #IMPLIED
                    %cellattrs;>
```

The **dimension** attribute defines the data; it can be used in spreadsheet calculations to link with related data in other matrices. For example, the dimension could be “size” for a dynamic column set derived from a list of sizes. If the same dimension is used elsewhere in a row or column set, the two sets can be used in linked calculations.

If the matrix has a `cellpfx` attribute, and a **dimension** is provided, a spreadsheet cell named “`PFX$DIMENSION`” will be set to the list of values from the set expression. Here `PFX` is the cell prefix and `DIMENSION` is the dimension. For example, if the `cellpfx` is “`szmat`” then the cell containing the size names in the example above would be `szmat$size`.

If the **required** attribute is true, the matrix will not be displayed if the set is empty. The initial implementation does not allow more than one `ColumnSet` per matrix, and does not support ‘sub columns’.

Currently the expression used to define the set should not contain direct or indirect references to other cells in this matrix.

Heading

The `Heading` element defines the heading for a matrix column.

```
<!ELEMENT Heading (#PCDATA)>
```

`PCData` represents parsed character data indicating that the element contains text and no other elements.

Example

```
<Column width="20" heading="Description"/>
```

Row

The `Row` element defines initial contents for a matrix row.

```
<!ELEMENT Row (Cell*)>
<!ATTLIST Row row          CDATA          #IMPLIED
              heading      CDATA          #IMPLIED
              headingexpr   (false | true) "false"
              map            CDATA          #IMPLIED>
```

The **row** attribute is the one-based row number.

The **heading** attribute is the row heading; it will be ignored if the `Matrix rowheadingwidth` attribute is not present.

If **headingexpr** is true, the heading is treated as an expression.

The **map** attribute defines a row-level mapping.

Example

```
<Row row="1" heading="false">
  <Cell col="1" editable="true" type="text" />
  <Cell col="2" editable="true" type="text" />
  <Cell col="3" editable="true" type="float" />
  <Cell col="4" editable="true" type="date" />
  <Cell col="5" editable="true" type="text" />
  <Cell col="6" editable="true" type="text" />
</Row>
```

RowSet

A dynamic row 'set' may be defined with the RowSet element.

```
<!ELEMENT RowSet (SubRow* | Cell*)>
<!ATTLIST RowSet row      CDATA      #IMPLIED
                  set      CDATA      #REQUIRED
                  dimension CDATA      #IMPLIED
                  required (false | true) "false"
                  map      CDATA      #IMPLIED>

<!ELEMENT SubRow (Cell*)>
<!ATTLIST SubRow heading CDATA #IMPLIED
                  map     CDATA #IMPLIED>
```

This contains a **set** attribute defining an expression which should evaluate to a list of row headings. A dynamic row set can also contain nested 'sub-row' SubRow elements. The subrows are numbered in the same sequence as normal rows.

For example:

```
<RowSet row="5" set="someexpr">
  <SubRow heading="sub1"/>
  <SubRow heading="sub2"/>
</Row>
```

This defines rows 5 and 6. Nested sub-rows are allowed with RowSets only.

The spreadsheet cell associated with a Cell in a RowSet will be an array; two-dimensional if the Cell is part of a dynamic column set.

See the notes on ColumnSet for a description of the dimension and required attributes.

A cell will be defined containing the row set list if the matrix has a cellpfx and the dimension is specified. For more details, see the ColumnSet notes.

The initial implementation does not allow more than one RowSet per matrix. A map attribute is not allowed if the RowSet contains sub-rows.

Currently, the expression used to define the set should contain direct or indirect references to other cells in this matrix.

Cell

The Cell element defines initial contents of a row/column cell and can also define a cell mapping.

```
<!ELEMENT Cell (#PCDATA | CellChoice | Value)*>
<!ATTLIST Cell col      CDATA      #REQUIRED
                  editable (true | false) "true"
                  heading (true | false) "false"
                  background CDATA      #IMPLIED
                  expr      CDATA      #IMPLIED
                  loglabel  CDATA      #IMPLIED
                  loglabelx CDATA      #IMPLIED
                  map      CDATA      #IMPLIED
                  %cellattrs;>
```

The **col** attribute is the 1-based column number. This attribute allows only the non-default Cell elements to be included.

The contents of the Cell are the initial value; the **editable** attribute defines whether the cell can be edited by the user.

If the **heading** attribute is true the cell contents are displayed differently, perhaps using a bold font.

The **expr** attribute defines a spreadsheet expression which is used to determine the cell contents. Any cell with an expression is implicitly non-editable.

An editable cell which has initial contents must be used with care. Once any edits have been made to the matrix in the client, the initial value will be saved to the server. The saved value will then continue to be used, even if the initial value is changed in the matrix definition.

If the cell type is 'choice', a CellChoice element may be present to define the choice items. There must be a CellChoice defined in the Cell, Column, or Matrix. Note that DTD rules require that the contents are surrounded by (*) indicating any number of CellChoice elements. However it is not legal to include more than one.

The initial value for a checkbox cell must be true or false; false is the default.

If there is an initial value, it should be defined by a single Value element; for compatibility with previous versions, the value can be defined directly in the Cell if there are no other elements. If there is a CellChoice element, the initial value is not allowed.

loglabel and **loglabelx** define the change log label for the cell. **loglabel** is a simple string, **loglabelx** is an expression evaluated in the context of the cell. **loglabel** and **loglabelx** cannot both be used.

Example

```
<Row row="1" heading="false">
  <Cell col="1" editable="true" type="text" />
  <Cell col="2" editable="true" type="text" />
  <Cell col="3" editable="true" type="float" />
  <Cell col="4" editable="true" type="date" />
  <Cell col="5" editable="true" type="text" />
  <Cell col="6" editable="true" type="text" />
</Row>
```

CellChoice

The CellChoice element defines the contents of a matrix 'choice' cell.<!ELEMENT Cell (#PCDATA | <!ELEMENT CellChoice (Option+)> <!ATTLIST CellChoice keyed (true | false) #FIXED "true">

CellChoice is similar to the basic Choice item, but has far fewer attributes. For clarity in this definition, and ease of parsing, a separate element is used. Note that all CellChoice elements are keyed.

CalcSet

CalcSet defines array calculations in a dynamic matrix.

```
<!ELEMENT CalcSet EMPTY>
<!ATTLIST CalcSet
  cell          CDATA          #REQUIRED
  expr          CDATA          #REQUIRED
  dimensions   CDATA          #REQUIRED
  flexible     (false | true) "false"
  map          CDATA          #IMPLIED>
```

The **dimensions** attribute lists the dimension values used to construct the array. It must contain one or two dimension strings, separated by commas, which match the dimensions of the ColumnSet or RowSet elements.

If the **flexible** attribute is true, the expression is evaluated with flexible dimension matching. This allows array values to be shared between matrices or spreadsheets when the dimensions do not match exactly.

Form

The form element defines a spec sheet form with input items.

```
<!ELEMENT Form (Defaults | TextField | IntField | FloatField | TextArea | Choice |
Image | Icon | Checkbox |
          SubForm | Label | MultiLabel | DateField | Custom | Matrix | Form
| Calc)*>

<!ATTLIST Form
  name          CDATA          #REQUIRED
  description   CDATA          #IMPLIED
  columns       CDATA          "1"
  tabbed        (false | true) "false"
  tabsinrow    CDATA          #IMPLIED
  vertical      (false | true) "false"
  scrollable    (false | true) "false"
  map          CDATA          #IMPLIED
  enabled       CDATA          #IMPLIED
  visible       CDATA          #IMPLIED
  background   CDATA          #IMPLIED
  border        (none | lined | etched) #IMPLIED
  perm         CDATA          #IMPLIED>
```

The mandatory **name** attribute defines the form name. The name is used to match the form against input data stored in the database.

The **description** attribute is used to generate a heading for the form.

The **columns** attribute defines the number of columns used in the display. The default is 1.

The **tabbed** attribute is true if the form items are displayed as separate tabs.

The **tabsinrow** attribute is relevant only if the form is tabbed; it defines the number of per row.

The **vertical** attribute selects an ‘old-style’ vertical set of textareas format.

Direct nesting of Matrix and Form elements is allowed without requiring a **SubForm**. A SubForm is needed only if attributes are required.

The **visible** attribute defines an expression which can be used to dynamically show or hide the form. The form will be visible if the expression evaluates to a non-zero number.

The **background** attribute sets the background color; if omitted the color is taken from the parent form or page.

The **border** attribute sets the border for the form.

Form level mapping is defined by the **map** attribute.

Example

In this specification sheet example, which incorporated SubForm elements, the **Cell** element is used to allow the user to enter contents to a field and map them to a miscellaneous field and mapped field.

```
<Form name="garment">
  <SubForm compat="true">
    <Form name="form1" columns="2" description="HEADER INFORMATION">
      <TextField label="Product Number" width="50" expr="style$stylenumber"/>
      <TextField label="Department" width="50" expr="style$department"/>
      <TextField label="Product Description" width="50" expr="style$shortname"/>
      <TextField label="Season" width="50" expr="style$season"/>
    </Form>
  </SubForm>

  <SubForm compat="true">
    <Form name="form1b" columns="2">
      <IntField label="Fabric Target" width="20" cell="style$misc_3"
map="fabrictgt"/>
      <TextField label="Construction Target" width="20" cell="style$misc_4"
map="constructtgt"/>
      <TextField label="Material Target" width="20" cell="style$misc_5"
map="materialtgt"/>
      <TextField label="Finish Target" width="20" cell="style$misc_6"
map="finishtgt"/>
    </Form>
  </SubForm>
</Form>
```

In this bid sheet example, the **expr** attribute is used to populate what was entered in the specification sheet's cell attribute and is linked to the bid sheet.

```
<SubForm compat="true">
  <Form name="form1c" columns="2" >
    <IntField label="Fabric Target" width="20" expr="style$misc_3"/>
    <IntField label="Fabric" width="20" map="fabric"/>
    <TextField label="Construction Target" width="20"
expr="style$misc_4"/>
    <TextField label="Construction" width="20" cell="style$misc_7"
map="construction"/>
    <TextField label="Material Target" width="20"
expr="style$misc_5"/>
    <TextField label="Material" width="20" cell="style$misc_8"
map="material"/>
    <TextField label="Finish Target" width="20" expr="style$misc_6"/>
    <TextField label="Finish" width="20" cell="style$misc_9"
map="finish"/>
  </Form>
</SubForm>
```

itemattrs

The itemattrs entity is shorthand for attributes shared by all items

```
<!ENTITY % itemattrs 'label          CDATA          #IMPLIED
                        width         CDATA          "0"
                        cellw         CDATA          #IMPLIED
                        cellh         CDATA          #IMPLIED
                        fill           (n | h | v | b)   #IMPLIED
                        pos            (c | n | ne | e | se | s | sw | w | nw) #IMPLIED
                        font           (b | i | bi)     #IMPLIED
                        fontsize      CDATA          #IMPLIED
                        labelw        CDATA          #IMPLIED
                        labelfont     (b | i | bi)     #IMPLIED
                        labelfontsize CDATA          #IMPLIED
                        labelalign    (l | c | r)      #IMPLIED
                        labelcolon    (false | true)  "true"
                        loglabel      CDATA          #IMPLIED
                        perm          CDATA          #IMPLIED
#IMPLIED'>
```

Attributes for some item types can also have a map attribute:

```
<!ENTITY % mapattrs '%itemattrs;
                    map CDATA #IMPLIED'>
```

Attributes for some item types can have a cell attribute:

```
<!ENTITY % itemcellattrs '%mapattrs;
                        index CDATA #IMPLIED
                        form  CDATA #IMPLIED
                        enabled CDATA #IMPLIED
                        cell   CDATA #IMPLIED'>
```

Attributes for item types can have expr and cell attributes:

```
<!ENTITY % itemexprattrs '%itemcellattrs;
                        expr CDATA #IMPLIED'>
```

The **compat** attribute is used with elements which do not have any associated data and which were supported in the old 'forms' language. These items were counted as part of the data store for the form, even though there could be no data. If an old form has existing data stored in the database, this attribute should be set to true to ensure that the column positions are preserved.

The compat attribute is not required with elements such as Defaults or Calc.

```
<!ENTITY % datalessattrs '%itemattrs;
                        compat (true | false) "false">
```

Example

The header row in this example uses font and width attributes:

```
<Form name="form4" columns="8">
<Defaults labelalign="r" fill="n" width="10" labelcolon="false"/>
<!-- main heading -->
<Label font="b" fontsize="4" value="Cost Calculation"/>
<Label cellw="6"/>
<Label font="b" fontsize="4" value="Please Review"/>
<!-- blank line-->
<Label cellw="8"/>
<Label cellw="4"/>
<Label font="b" fontsize="2" cellw="2" value="First Cost"/>
<Label font="b" fontsize="2" cellw="1" value="Quantity"/>
<Label font="b" fontsize="2" cellw="1" value="Extended Cost"/>
```

In this example, the Fabric Cost in the Bill of Materials table does not have a field label for the log file to use, so we defined one.

```
<IntField cell="fabriccost" map="fabriccost" loglabel="Fabric Cost"/>
```

In this example, note that all these items are mapped to an attribute:

```
<IntField label="Fabric Target" width="20" cell="style$misc_3" map="fabrictgt"/>
<TextField label="Construction Target" width="20" cell="style$misc_4"
map="constructtgt"/>
<TextField label="Material Target" width="20" cell="style$misc_5"
map="materialtgt"/>
<TextField label="Finish Target" width="20" cell="style$misc_6" map="finishtgt"/>
```

In these examples, the items are set using the cell attribute and then how they appear linked using the expr attribute:

- In this specification sheet example, the cell element is used to allow the user to enter contents to a field and map them to a miscellaneous field and mapped field.

```
<IntField label="Fabric Target" width="20" cell="style$misc_3"
map="fabrictgt"/>
<TextField label="Construction Target" width="20" cell="style$misc_4"
map="constructtgt"/>
<TextField label="Material Target" width="20" cell="style$misc_5"
map="materialtgt"/>
```

- In this bid sheet example, the expr element is used to link the contents of the field in the specification sheet to the bid sheet. The user is not able to enter information in the bid sheet because expr prevents editing.

```
<IntField label="Fabric Target" width="20" expr="style$misc_3"/> (LINKED FROM SPEC SHEET)
<TextField label="Construction Target" width="20" expr="style$misc_4"/> (LINKED FROM SPEC SHEET)
<TextField label="Material Target" width="20" expr="style$misc_5"/> (LINKED FROM SPEC SHEET)
```

Defaults

The Defaults element sets defaults for the basic item attributes.

```
<!ELEMENT Defaults EMPTY>
<!ATTLIST Defaults %itemattrs;>
```

Example

```
<Defaults labelalign="r" fill="n" width="10" labelcolon="false"/>
```

Common Initial Value

Common “initial value” element. With some elements, a Value attribute can be used as an alternative. The value element and attribute must not be used together.

```
<!ELEMENT Value (#PCDATA)>
```

Simple Item Types – TextField and IntField

Simple item types can have the default value as an attribute.

```
<!ELEMENT TextField (Value?)>
<!ATTLIST TextField %itemexprattrs;
                limit CDATA #IMPLIED
                value CDATA #IMPLIED>

<!ELEMENT IntField (Value?)>
<!ATTLIST IntField %itemexprattrs;
                value CDATA #IMPLIED>
```

Example

```
<TextField label="Construction Target" width="20" cell="style$misc_4"
map="constructtgt"/>
<IntField label="Fabric Target" width="20" cell="style$misc_3" map="fabrictgt"/>
```

FloatField

The FloatField element defines a floating point number.

```
<!ELEMENT FloatField (Value?)>
<!ATTLIST FloatField %itemexprattrs;
                value CDATA #IMPLIED
                prec CDATA #IMPLIED>
```

The **prec** attribute defines the number of decimal places; the default is 2.

Example

This example shows a number of the FloatField labels that are currently defined within the bid sheet:

```
<FloatField label="First Cost $" width="20" cell="style$misc_20" map="firstcost"
prec="3"/>
<FloatField label="Packaging Cost $" width="20" map="packcost"
cell="style$misc_21" prec="3"/>

<FloatField label="*Cost & Packaging Total $" width="20" cell="style$itemcost"
expr="zsum(style$misc_20,style$misc_21)" prec="3"/>
```

TextArea

The TextArea element can be used to define spaces for comments or other text information.

```
<!ELEMENT TextArea (Value?)>
<!ATTLIST TextArea %itemexprattrs;
    rows CDATA #REQUIRED>
```

The **rows** attribute for a text area is the number of text rows

Example

This example shows how the comments fields can be defined as TextAreas:

```
<Form name="form10" description="Comments">
<TextArea label="Retailer Comments" width="50" cell="style$misc_60" rows="4"
map="retailercomm"/>
<TextArea label="Supplier Comments" width="50" cell="style$misc_61" rows="4"
map="suppcomm" perm="p1,p2"/>
...
</Form>
```

Icon

The Icon element defines an image from icon list. In order for icons to appear, they must be uploaded via Spectrum – Icons.

```
<!ELEMENT Icon EMPTY>
<!ATTLIST Icon %datalessattrs;
    iconname CDATA #REQUIRED
    dim CDATA #IMPLIED>
```

The **iconname** attribute defines the name of the icon. The **dim** attribute defines the display size of the image; if omitted the actual image size is used.

dim defines the image dimensions in pixels by width and height and must be WxH, where W and H are integers.

Example

```
<Form name="logo">
    <Icon compat="true" iconname="samplelogo"/>
</Form>

    <Image label="1" compat="true" dim="80x80" cellh="5"/>
<ImageOption name="Constructionimage1"/>
</Image>
```

Checkbox

Checkbox items have a **state** attribute to set the initial value.

```
<!ELEMENT Checkbox EMPTY>
<!ATTLIST Checkbox %itemcellattrs;
                state (false | true) "false"
                group CDATA      #IMPLIED>
```

A checkbox with a group attribute acts like a 'radio' button. Only one checkbox in the group is set at a time.

Example

```
<Form columns="2" description="Inner Packing" name="form6">
<Checkbox label="Boxes"/>
<Checkbox label="Eggcrate Carton"/>
<Checkbox label="Crosscrated Carton"/>
<Checkbox label="Inner Carton"/>
<Checkbox label="Master Polybag"/>
</Form>
```

Label

The Label element defines a single-line label.

```
<!ELEMENT Label (Value?)>
<!ATTLIST Label %datalessattrs;
                align (l | r | c) #IMPLIED
                value CDATA      #IMPLIED>
```

Example

```
<TextField label="Division" width="20"/>
<TextField label="Department" width="20"/>
<TextField label="Product" width="20"/>
<TextField label="Description" width="20"/>
<TextField label="Class" width="20"/>
<Label compat="true"/>
```

MultiLabel

MultiLabel defines a multi-line wrapped label.

```
<!ELEMENT MultiLabel (Value?)>
<!ATTLIST MultiLabel %datalessattrs;
                dim CDATA #IMPLIED>
```

The **dim** attribute defines the display width in pixels. The default is 300.

DateField

Date entry field.

```
<!ELEMENT DateField (Value?)>
<!ATTLIST DateField %itemexprattrs;
            value CDATA #IMPLIED
            mode (std | local | time) "std"
            icon (true | false) #IMPLIED>
```

The **mode** attribute defines the operation and value returned. It must be one of the following:

- **std** – the date defined using “standard” server time zone
- **local** – the date defined using local time zone
- **time** – the date & time defined using local time zone.

The default is **std**.

Example

```
<DateField label="Created Date" mode="local"/>
```

Custom

Custom item requires a **name** attribute to define the item.

```
<!ELEMENT Custom (Parameter*, Value?)>
<!ATTLIST Custom %itemexprattrs;
            name CDATA #REQUIRED
            imageid CDATA #IMPLIED>
```

It can have nested parameter elements to provide further information to the component creator.

Parameter

A Parameter defines a single extra configuration parameter for a custom component.

```
<!ELEMENT Parameter (#PCDATA)>
<!ATTLIST Parameter name CDATA #REQUIRED>
```

SubForm

The Subform item encloses a nested form or matrix.

If the nested item is omitted, a name must be present referring to an earlier form. These shared forms must not contain, directly or indirectly, any data entry items.

```
<!ELEMENT SubForm ((Matrix | Form)?)>
<!ATTLIST SubForm %datalessattrs;
          name CDATA #IMPLIED>
```

Example

```
<Form columns="3" name="form6c">
  <SubForm compat="true">
    <Form description="MASTER" name="form6c1">
      <TextField label="Units per CTN" width="10"/>
      <TextField label="Net Weight (kgs)" width="10"/>
      <TextField label="Gross Weight (kgs)" width="10"/>
      <TextField label="Length (cms)" width="10"/>
      <TextField label="Width (cms)" width="10"/>
      <TextField label="Heigh (cms), " width="10"/>
      <TextField label="Volume" width="10"/>
    </Form>
  </SubForm>
  <SubForm compat="true">
    <Form description="INNER" name="form6c2">
      <TextField label="Units per Inner" width="10"/>
      <TextField label="Net Weight (kgs)" width="10"/>
      <TextField label="Gross Weight (kgs)" width="10"/>
      <TextField label="Length (cms)" width="10"/>
      <TextField label="Width (cms)" width="10"/>
      <TextField label="Heigh (cms), " width="10"/>
      <TextField label="Volume" width="10"/>
    </Form>
  </SubForm>
  <SubForm compat="true">
    <Form description="LOADING" name="form6c3">
      <TextField label="Units/20'" width="10"/>
      <TextField label="Unit/40'" width="10"/>
      <TextField label="Units/HC" width="10"/>
      <TextField label="Vol per 20' Container" width="10"/>
      <TextField label="Vol per 40' Container" width="10"/>
      <TextField label="Vol per HC Container" width="10"/>
    </Form>
  </SubForm>
</Form>
```

Choice

The Choice element contains a number of options. Each option element contains the displayed choice and an optional value that is used for external export and updates. The first item is the default, unless an option is present with the selected attribute set to true.

```
<!ELEMENT Choice (Option+)>
<!ATTLIST Choice %itemcellattrs;
              keyed (true | false) "false">
```

By default, the value stored in the database for a Choice item is the index of the selected option. This means that if options are added or the list is reordered, the stored value may refer to a different option.

To avoid this scenario, a Choice item can be **keyed**. In this case each option must include a distinct key attribute. The key is stored in the database, allowing option lists to be changed without disturbing the value.

Example

Below is an example of the Choice field that will appear as a drop-down list box within the bid sheet.

```
<Choice label="Show Country of Origin on Label">
  <Option>Yes</Option>
  <Option>No</Option>
</Choice>
```

Option

The Options element defines items in a Choice.

```
<!ELEMENT Option (#PCDATA)>
<!ATTLIST Option value CDATA #IMPLIED
                 selected (true | false) "false"
                 key CDATA #IMPLIED
                 list CDATA #IMPLIED
                 expr CDATA #IMPLIED>
```

The **key** attribute is compulsory, if the Choice is keyed.

The **list** attribute defines a lookup parameter code which is used to get the real options for this item. All the active (and current) values for the parameter are included. If there is a default value for the parameter, it is included as the first item; the remaining items are alpha sorted. If list is used, the option contents must be empty.

If a list option is marked as the default (with selected = true), the default item is the first in the list.

The **expr** attribute defines a spreadsheet expression, which evaluates to the real options for this item. expr is similar to list except that the set of values can depend on other values in the spreadsheet.

The value attribute defined the string used for import and export processes to refer to the option. If value is not present, the string itself is used. The value attribute is ignored if a value list is set.

Example

```
<Choice label="Fabric Mills" keyed="true" width="20" cell="style$misc_100"
map="fabricmill">
<Option selected="true" key="x" value="">&lt;Select&gt;</Option>
<Option list="fabricmilllist" key="y"/>
</Choice>
```

(Also see Choice examples.)

Image

The Image item contains either a single name or a set of named options.

```
<!ELEMENT Image (ImageOption+)>
<!ATTLIST Image %datalessattrs;
          dim CDATA #IMPLIED
          compact (true | false) "false">
<!ELEMENT ImageOption (#PCDATA)>
<!ATTLIST ImageOption name CDATA #REQUIRED>
```

The **dim** attribute defines the display size of the image in pixels by width and height and must be WxH, where W and H are integers. If omitted, fixed defaults are used.

If **compact** is 'true', the display does not reveal the notes, change and fullsize buttons; instead a pop-up menu is available for these functions.

Example

Below is an example of how an image (or three) could be represented within the specification sheet language.

```
<Form name="Packaging Images" columns="3" description="Packaging Image">
<Image label="Packaging #1" compat="true" dim="210x210" cellw="1">
<ImageOption name="pack1"/>
</Image>
<Image label="Packaging #2" compat="true" dim="210x210" cellw="1">
<ImageOption name="pack2"/>
</Image>
<Image label="Packaging #3" compat="true" dim="210x210" cellw="1">
<ImageOption name="pack3"/>
</Image>
```

Calc

The Calc element defines a spreadsheet cell with associated expression. Both cell name and expression are required attributes.

```
<!ELEMENT Calc EMPTY>
<!ATTLIST Calc cell CDATA #REQUIRED
          expr CDATA #REQUIRED
          function (true | false) "false"
          map CDATA #IMPLIED>
```

If the **function** attribute is true, the element defines a spreadsheet function; the cell name is the function name and the expression is the body.

The **map** attribute defines a mapping for the calculated value. map cannot be used with function declarations.

The Calc element is not associated with any form of GUI component.

Example

Below is an example of the functions represented in the soft home ELC sheet:

```
<Calc cell="commission" expr="lookupdate('commission', style$orderby, style$agent)"/>
```

```
<Calc cell="commissionpercent" expr="commission/100"/>
```

```
<Calc cell="commissioncalc" expr="($1 * $2)" function="true"/>
```

This example shows how the commission would actually be calculated using the function from above>

```
<FloatField labelfontsize="2" labelfont="b" cell="style$misc_240" cellw="1" expr="commissioncalc(firstcost, commissionpercent,)" map="commissioncost" prec="3"/>
```

XFO Templates

Oracle Retail Design supports the generation of printable output on the client and server side. The client side printing is generated based on specified rules within the code. If users are expecting an exact format, they may not choose to leverage the client side print. The server side printing leverages specific format files that have been defined and uploaded within Oracle Retail Design. The server side printing is available from within the client-side printing option dialog box and is also used during the technical specification export process supported by Oracle Retail Integrator. Format files can be developed using XFO technology and can be uploaded by browsing to the following URL from within the Oracle Retail Design Administration Console:

- <https://www.retail.com/applications/design/template.jsp>

The administrator is prompted to browse and upload the template file, define the format that it uses and identify a mode that will be used to cross-reference the configuration file. The template mode is a free-form text field and the value input is used to cross-reference the uploaded file in other configuration administration steps. Specifically, the mode is used within the enterprise and user view configuration file to identify a format that can be used by the server side printing options available to the user. In addition, the mode can be referenced within Oracle Retail Integrator as the `ObjectiveSheetType` during the setup of the run type used to support the technical specification export process.

Note: Although the objective sheet template upload process continues to support the upload of .xmf files, Oracle Retail recommends that xfo formats be used.

Objective Sheet template upload

File:

Template type:

Template mode:

Browse to required file, and press **Upload** to transmit file to server.

Enter a template *mode* when uploading templates for special situations, including style file reporting. The mode should consist of letters, digits and period characters.

Print Template Upload Window in Oracle Retail Design

The actual development of the XFO files can be done in any text editor. A text editor that supports XML would be most efficient and is recommended. This chapter focuses on the primary functions supported within the development of the XFO templates.

XFO Introduction

The template-driven PDF generator used to format print files on the Oracle Retail Design client and within the technical specification export process is named 'XFO'. An XFO template is an XML file containing a mixture of markup XML and Style File XFO processing elements which are used to control the output and to include dynamic values.

The first implementation uses the XSL formatting objects (XSL-FO) markup language, in conjunction with Apache FOP (<http://xml.apache.org/fop>) which is a XSL-FO to PDF renderer. FOP implements most of the XSL-FO standard, but there are some limitations. See documents on the website for conformance details.

Template names have the form **xfo-[mode-]E**, where **E** is the enterprise ID and **mode** is the optional mode string. The engine searches for a template with the correct enterprise first, then tries a file with enterprise 0. This design allows default templates for common requirements such as comments printing.

XFO Operation

The XFO processor first reads and parses the template file. Any XML errors found at this stage will be reported by an error PDF produced by the processor. When a PDF is generated from a set of styles, the 'style file' elements are processed to produce pure XSL-FO output, which is passed directly to the FOP engine for rendering to PDF.

Basic Structure

An XFO template will contain elements from the XSL-FO namespace and the 'style file xfo' control namespace. Conventionally prefixes **fo:** and **sf:** are used for these namespaces.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
        xmlns:sf="http://www.retail.com/XSL/style-files">
```

The namespace URLs should be included exactly as above. If you need to embed SVG for advanced graphics, add the SVG namespace to the root element or the SVG element when used:

```
<svg:svg xmlns:svg="http://www.w3.org/2000/svg">
```

Expressions and Attributes

Attributes in **fo:** (and **svg:**) elements may contain expressions enclosed in `{` and `}`. These expressions are evaluated during the processing phase and final attribute value is passed to the XSL-FO output. Expressions are written using the standard 'spec sheet' expression language and may refer to variables set earlier in the processor.

Example:

```
<fo:simple-page-master master-name="one"
  page-height="{pageheight}{unit}"
  page-width="{pagewidth}{unit}">
```

```
<fo:block break-before="{index > 0 ? 'page' : 'auto'}">
```

Here the page dimensions in the page master as set using previously defined numbers and a unit string (mm, cm, in, and so on). In the block element, the value of the break-before attribute is set to page or auto according to the value of the index variable.

Note that in common with all XML files, any `&` or `<` characters in expressions must be written by the character entities `&` and `<`;

SF Processing Elements

The following **sf:** processing elements are available.

Attributes which are *expressions* are evaluated directly. They are *not* enclosed in `{ }`. An attribute defined as a *string* may contain `{ }` expressions.

An attribute defined as a *name* represents a ‘variable’ name. It must follow the rules for Java identifiers. Essentially, the first character must be a letter or `_` and the remainder can be letters, digits, or `_`.

Any **sf:** element which contains other elements defines a new ‘context,’ variables defined in this context are not in scope outside it.

sf:str

```
<sf:str x="expression" />
```

Evaluate the expression and include the result as a string at the current point in the XSL-FO output.

Example

```
<fo:block><sf:str x="item -> shortname" /></fo:block>
```

sf:int

```
<sf:int x="expression" />
```

Evaluate the expression and include the result as an integer at the current point in the XSL-FO output.

Example

```
<fo:block><sf:int x="item -> quantity" /></fo:block>
```

sf:float

```
<sf:float x="expression" fmt="string" />
```

Evaluate the expression and include the result as a decimal number at the current point in the XSL-FO output. The optional `fmt` attribute can be used to supply a format for the conversion (see [java.text.DecimalFormat](#)); if omitted a default format with 2 decimal places is used.

Example

```
<fo:block><sf:float x="item -> elctarget" fmt="0.000" /></fo:block>
```

sf:date

```
<sf:date x="expression" fmt="string" tz="string" />
```

Evaluate the expression as in internal date value (a Java time stamp divided by 1000), convert the date to a string using the supplied format or a default, and include the result in the XSL-FO output. See [java.text.SimpleDateFormat](#) for details of the optional format string. If the format is omitted a locale-specific date format is used.

The optional `tz` attribute is used to select the time zone for the formatting. It must be one of the time zone IDs understood by `java.util.TimeZone` (for example “Europe/London” or “PST”). If `tz` is omitted, the server’s ‘standard’ time zone is used. Most dates in design (initial availability, bid deadline, and so on) represent a ‘day’ and are stored internally using the standard time zone; `tz` should not be used with these. Dates which represent a point in time (last change, log dates, and so on) contain a time of day element and the time zone is relevant.

Examples

```
<sf:date x="item -> biddeadline"/>
<sf:date x="item -> changedate" tz="{tz}" fmt="yyyy-MM-dd HH:mm:ss"/>
```

The first example displays a date using the default format; the second displays a time stamp using such as 2004-11-09 12:23:13 with a timezone obtained from a variable.

sf:set

```
<sf:set n="name" x="expression"/>
or
<sf:set n="name" x="expression">
  ... content ...
</sf:set>
```

The expression is evaluated and assigned to the name. In the first syntax, the name remains in scope until the end of the current context. In the second example, the name is in scope during the processing of the embedded content.

Example

```
<sf:set n="unit" x="'mm'"/>
<sf:set n="temp" x="x * 10">
  ... content ...
</sf:set>
```

The first example sets `unit` to the string “mm”; the second defines `temp` for the processing of the enclosed content.

sf:update

```
<sf:update n="name" x="expression"/>
or
<sf:update n="name" x="expression" index="expression"/>
```

The expression is evaluated and assigned to the most recent definition of the name. The name should have been defined by an earlier `sf_set` element. In the second form, the `index` expression is evaluated and used to set an element in the array identified by the name. The array should have been created earlier using the `array` function.

Examples

```
<sf:update n="count" x="count+1"/>
Set count to its previous value, plus 1.
<sf:set n="arr" x="array(10)"/>
...
<sf:update n="arr" x="1" index="i+1"/>
```

Set element `i+1` in the array `arr` to 1.

sf:func

```
<sf:func n="name" x="expression" />
```

or

```
<sf:func n="name" x="expression">
  ... content ...
</sf:func>
```

Define the expression as a function. The rules for the scope of the name are as in **sf:set**.

Example

```
<sf:func n="imwidth" x="$1 * min(1, min(maxwidth/$1, maxheight/$2))" />
```

sf:if

```
<sf:if x="expression">
  ... content ...
</sf:if>
```

If the expression is defined and non-zero, then process the content; otherwise, ignore the content.

Alternative Format

```
<sf:if x="expression">
  <sf:then>
    ... content 1 ...
  </sf:then>
  <sf:else>
    ... content 2 ...
  </sf:else>
</sf:if>
```

In this form, the content in the ‘then’ part is processed if the expression is non-zero; otherwise the content in the ‘else’ part is processed. The ‘else’ part can be omitted but that is the same as the more succinct first format.

sf:for

```
<sf:for n="name" to="expression" from="expression" by="expression"
  while="expression" set="expression" var="name">
  ... content ...
</sf:for>
```

sf:for is used to process content repeatedly. All the attributes are optional, but at least one of two, `while` or `set` must be used. To prevent the server running for ever as a result of a faulty template, a limit of 8192 iterations is imposed by the processor.

There are three distinct forms of iteration; any combination may be used:

- Numeric

Iterate over the range ‘from’ to ‘to’ inclusive, in steps of ‘by’. If `from` is omitted, the iteration starts at 1; `by` is omitted the step is 1. If the name `n` is supplied, the iteration value is assigned to it during the loop.

This is roughly equivalent to the java loop:

```
for (name = from; name <= to; name += by)
```

Except that if the step `by` is negative, the loop counts down and the test is `name >= to`.

- Conditional

If `while` is used, the loop terminates as soon as the expression evaluates to ‘false’ (undefined or zero).

- Set

The set expression should evaluate to a set of items; the loop continues whilst there are elements in the set; the current item is assigned to the name defined by `var`, if present. The item set will be defined outside the processor.

Examples

```
<sf:for to="10">
  .. content ..
</sf:for>
```

Process the content 10 times.

```
<sf:for n="index" from="0" set="items" var="item">
  ... content ...
</sf:for>
```

Process the content over the set of items; the current item is assigned to `item`; the variable `index` counts up from zero.

sf:macro

```
<sf:macro n="name">
  ... content ...
</sf:macro>
```

Store the content against the name for later use. The macro is expanded using the **sf:call** element. Macros are useful for repeated header components, and so on.

sf:call

```
<sf:call n="name">
  <sf:set n="name1" x="expression1"/>
  <sf:set n="name2" x="expression2"/>
  ...
</sf:call>
```

Process the content of the macro name; while processing `name1`, `name2...` are set to `expression1`, `expression2...`

The nested `sf:set` elements are optional.

Builtin Values and Functions

The processor always defines the variable `now` as the current date. This can be used to include the time of printing in a footer, for example:

```
<fo:block font-size="5pt">
  <sf:date x="now" fmt="yyyy-MM-dd HH:mm:ss" tz="Europe/London"/>
</fo:block>
```

The following functions are always available:

array(n)

Create a 1-dimensional array of size `n`. The size must not be more than 512.

Example:

```
<sf:set n="arr" x="array(size+1)"/>
```

geticon(string)

Lookup the 'icon' named by the argument. The result is *undefined* if the icon does not exist, and an icon object otherwise:

| Field | Type | Meaning |
|--------|---------|-----------------|
| file | string | Image file name |
| width | integer | Image width |
| height | integer | Image height |

Example:

```
<fo:external-graphic src="url({geticon('logo') -> file})"/>
```

getprop(prop) or getprop(prop, deflt)

Lookup a *property* passed to the processor. If the property is not defined, the result is *undefined* or **deflt** if there are two arguments.

Example:

```
<sf:set n="pagewidth" x="getprop('pagewidth', 210)"/>
```

valuekey(v)

A value which is derived from a parameter lookup (for example the value of a mapped dropdown list box) may have an associated *external value*. This function returns the external value of v, if any, or v if there is no external value.

hasmorevalues(set)

The argument must evaluate to a 'set' (see sf:for); return 1 if there are further items in the set or zero if the set is exhausted. This can be used to determine whether a break is needed after an item.

XFO and Styles

When processing Oracle Retail Design styles, there are additional predefined values and functions.

Values

local

`local` is set to 1 if the user generating the output is in the same enterprise as the style owner, and 0 if the user is in a different enterprise. This matches the `style$local` value in spec sheets and can be used to produce different output for local and partner users.

When creating objective sheets for tech spec output, `local` is set to 0.

itemcount

The number of style items being processed. This should not be used to iterate over the set. Use `items` (below). The count may be used to estimate layouts, and so on but is not guaranteed to be accurate. It is possible that one or more of the styles selected by the user have been deleted by another user between selection and processing.

items

`items` is a set of style objects. Iterate over the set with an `sf:for` element (see above).

Each object contains the fields show below. Boolean fields are represented as a *combined* value with a numeric par of 0 or 1 and a string part of “false” or “true.”

Dimension objects have integer `width` and `height` fields.

| Field | Type | Meaning |
|--------------------|-----------|-----------------------------------|
| designid | string | Style number |
| retailername | string | Name of retailer enterprise |
| shortname | string | Style short name |
| longname | string | Style long name |
| creationdate | date | Creation date |
| changedate | date | Last change date |
| lastedituser | user | User who last edited style |
| lasteditentname | string | Enterprise name of last edit user |
| datachange | boolean | Data change flag |
| documentchange | boolean | Document change flag |
| documentexportdate | date | Date of last document export |
| division | hierarchy | Style division |
| department | hierarchy | Department |
| clazz | hierarchy | Class |
| subclazz | hierarchy | Subclass |

| Field | Type | Meaning |
|---------------------|-----------|---------------------------------|
| buyer | user | Buyer |
| quality | user | Quality manager |
| designer | user | Designer (in retailer) |
| availdate | date | Initial availability |
| biddeadline | date | Bid deadline |
| deliveries | integer | # of deliveries |
| text1 | string | Text field 1 (features) |
| text2 | string | Text field 2 (comments) |
| text3 | string | Text field 3 (proposed changes) |
| category | string | Category |
| dispatchtype | string | Dispatch type |
| supplierid | integer | ID of supplier |
| suppliername | string | Name of supplier enterprise |
| suppliernum | string | Supplier number (in retailer) |
| supplieraccountnum | string | Supplier account number |
| supplierproductnum | string | Supplier design ID |
| accountmanager | user | Account manager |
| supplierdesigner | user | Designer in supplier |
| suppliertechologist | user | Technologist in supplier |
| supplierdepartment | hierarchy | Supplier department (team) |
| countryid | integer | ID of country (COO) |
| countrycode | string | Country code |
| countryname | string | Country name |
| locationid | integer | ID of location (factory) |
| locationname | string | Location name |
| locationcode | string | Location code |
| fobcountryid | integer | ID of FOB country |
| fobcountrycode | string | FOB country code |
| fobcountryname | string | FOB country name |
| foblocationid | integer | ID of FOB location |
| foblocationname | string | FOB location name |
| foblocationcode | string | FOB location code |

| Field | Type | Meaning |
|-----------------|---------------|----------------------------|
| agentid | integer | ID of agent |
| agentname | string | Name of agent enterprise |
| agentnum | string | Agent number (in retailer) |
| agentaccountnum | string | Agent account number |
| agentcontact | user | Agent contact |
| priceoption | integer | Price by option |
| costoption | integer | Cost by option |
| quantity | integer | Total volume |
| quantityoffered | integer | Supplier volume |
| retail | decimal | Retail price |
| cost | decimal | Item code |
| vat | decimal | VAT |
| netcost | decimal | Net cost |
| margin | decimal | Buying margin |
| elctarget | decimal | ELC target |
| leadtime | integer | Lead days |
| orderreqdby | date | Order required by |
| effectiveuntil | date | Effective until |
| seasonid | integer | ID of season |
| seasoncode | string | Season code |
| seasonname | string | Season name |
| seasonstart | date | Season start |
| seasonend | date | Season end |
| phasecode | string | Phase code |
| phasename | string | Phase name |
| themename | string | Theme name |
| retailerstatus | status | Retailer status |
| supplierstatus | status | Supplier status |
| typename | string | Product type name |
| type | integer | ID of product type |
| typecodes | integer array | Spec and bid sheet codes |
| elctypename | string | ELC type name |

| Field | Type | Meaning |
|---------------|--------------------|----------------------------|
| elctype | integer | ID of ELC type |
| eltypecodes | integer array | ELC sheet code (element 0) |
| sizerange | sizerange | Size range |
| colours | colour array | Colours |
| labels | label array | Labels |
| volumes | integer array (2D) | Volumes by size and colour |
| volumestotal | integer | Total computed volumes |
| sizevolumes | integer array | Volumes by size |
| colourratios | decimal array | Colour ratios |
| colourvolumes | integer array | Volumes by colour |
| images | attachment array | Summary and spec images |
| documents | attachment array | Attached documents |

ID values (such as *seasonid*) can be used for parameter lookups. They are internal values and have no external meaning.

Subsidiary Objects

User

The value of a **user** object is the user's name. The object also contains these fields:

| Field | Type | Meaning |
|-------|--------|----------------------|
| name | string | User's name |
| email | string | User's email address |

Hierarchy

The value of a **hierarchy** object a combined value with the hierarchy item internal ID as the numeric part, and the name of the item as the string part. The object also contains these fields:

| Field | Type | Meaning |
|-------|---------|-------------------------------|
| id | integer | Internal ID |
| name | string | Name of hierarchy item |
| code | string | Number/code of hierarchy item |

Status

A **status** object contains the following fields:

| Field | Type | Meaning |
|-------|--------|-----------------------|
| name | string | State name |
| date | date | Date of status change |

Sizerange

A **sizerange** object contains the following fields:

| Field | Type | Meaning |
|-------|----------|----------------------|
| name | string | Size range name |
| code | string | Size range code |
| list | sizelist | List of actual sizes |

Sizelist

A **sizelist** object contains the following fields:

| Field | Type | Meaning |
|---------|------------|-------------------------------------|
| columns | integer | Number of columns in range (1 or 2) |
| sizes | size array | Array of individual sizes |

Size

A **size** object contains the following fields:

| Field | Type | Meaning |
|---------|---------|--|
| name | string | Size name |
| code | string | Size code |
| percent | decimal | Size percentage from size range definition |

To iterate over all the sizes in a style, use elements like:

```
<sf:set n="sizes" x="item -> sizerange -> list -> sizes"/>

<sf:for n="s" from="0" to="length(sizes)-1">
  <sf:set n="size" x="sizes[s]"/>
  ...
  <sf:str x="size -> name"/>
  ...
</sf:for>
```

Colour

A **colour** object contains the following fields:

| Field | Type | Meaning |
|-------|---------|-----------------------|
| id | integer | Internal ID of colour |
| name | string | Colour name |
| code | string | Colour code |
| rgb | integer | Colour RGB value |

Label

A **label** object has the following fields:

| Field | Type | Meaning |
|-------------|--------|--------------------------------|
| comment | string | Label comment |
| whenadded | date | Date when label added to style |
| code | string | Label template code |
| description | string | Label template description |
| typename | string | Label template type |
| subtypename | string | Label template subtype |

Attachment

An **attachment** object has the following fields:

| Field | Type | Meaning |
|--------------|------------------|---|
| file | string | File name on server of image or document |
| internalname | string | Attachment 'internal name' |
| name | string | Attachment name |
| size | dimension | Image size; not set for documents |
| date | date | Upload/import date of image or document |
| caption | string | Annotated image caption |
| annotations | annotation array | Array of image annotations |
| styleimage | boolean | 1 if the image is from the 'summary' images tab, zero otherwise |

The internalname field indicates the location of the image in the style (for the summary images tabs the value is 0, 1, 2... for each tab). The simplest way to obtain a specific image is via the findimage function.

The name field is useful for documents. It is the name as shown in the documents list on the summary tab.

The size field is a dimension object with integer width and height fields. The caption and annotations fields are set from the style image notes view; the caption is the text area at the bottom and the annotations are the text notes.

Annotation

An **annotation** object contains the following fields:

| Field | Type | Meaning |
|------------------|-----------|--|
| text | string | Annotation text |
| foregroundcolour | integer | RGB value of text colour (may be unset) |
| backgroundcolour | integer | RGB value of background colour (may be unset) |
| position | fraction | Position of annotation; the x, y values give the position as fractions of the display area for the image |
| size | dimension | Size of text box in pixels |
| pointer | fraction | Position of pointer end within image; the x, y values give the position as fractions of the image size (for example, [0.5, 0.5] represents the centre of the image). |

A **fraction** object contains decimal **x** and **y** fields. It represents a position as a fraction of another dimension.

Standard **sf:** and **fo:** elements in conjunction with SVG graphics can be used to draw images and the annotations. See the example template.

Functions

There are additional functions available to obtain more complex style values. As well as the functions listed below, the standard parameter lookup and formatting functions are also available. Note that mapped values obtained from **getmap** represent the ‘export’ forms of values rather than the internal forms. In particular values associated with spec sheet <Choice> items will be represented using the option “value” or parameter “external value.” Such values will not behave in the same way if used in arguments to lookup functions. If there is any doubt, calculate values with <Calc> elements in the spec sheet.

getmisc(item, name)

Get the misc value name from the item (the first argument must be a style item). If the name is a number or a string starting with a digit, “misc_” is prefixed automatically.

Example:

```
getmisc(item, 100)
Get misc value 100.
```

getlink(item, name)

Get the link value name from the item (the first argument must be a style item). If the name is a number or a string which does not start with “link_”, then “link_” is prefixed automatically.

Example:

```
getlink(item, 'mkSE')
```

Get linkage value “link_mkSE”.

getmap(item, tab, name)

Get a mapped value named *name*. The first argument is a style item and the second defines the tab in the usual way – *spec*, *bid*, *elc* or *tab:key*. If the mapped item refers to a form, matrix, row or column the result will be an array.

findimage(item)**findimage(item, X)**

Find an **attachment** object for a style image. In the single argument form, the first image from the summary image tabs is returned. In the second format, image **X** is returned (**X** is “0”, “1”, “2” ...).

If the indicated image is not present on the style, the result is *undefined*.

findimage(item, tab, formname, imagename)

Find an image from a spec sheet. The second argument indicates the spec tab (as for **getmap**); the third argument is the form name on the spec sheet and the final argument indicates the image name in the spec sheet <Image> item.

Example:

```
<sf:set n="ig" x="findimage(item, 'spec', 'form3', 'img1')"/>
```

Find an image in form “form3” on the specification tab of the item.

getchangelog(item, maxnum, days, roles)

Return change log records for the style. The first argument is a style item; the remaining arguments are optional.

maxnum is the maximum number of records returned; if zero or omitted there is no limit.

days specifies the maximum number of days back to go when loading records; if zero or omitted there is no limit (for example, specifying 7 would return all changes in the past week).

roles sets ‘security role keys’ to control the changes that are returned. If omitted, the default setting is ‘partner users’.

The result of **getchangelog** is an array of change records, each of which has the following fields:

| Field | Type | Meaning |
|----------|--------|--|
| user | string | Name of user making change |
| date | date | Date/time of change |
| comment | string | Change comment (create, edit, import, and so on) |
| field | string | Name of field which is changed |
| previous | string | Previous value |
| current | string | New value |

For example:

```
<sf:set n="log" x="getchangelog(item, 100)">
  <sf:if x="length(log) > 0">
    ... define table for log display ...

    <sf:for n="ixl" from="0" to="length(log)-1">
      <sf:set n="rec" x="log[ixl]"/>
    ...
```

getcomments(item, maxnum, days)

Return style file comment records for the style. The first argument is a style item; the remaining arguments are optional.

maxnum is the maximum number of records returned; if zero or omitted there is no limit.

days specifies the maximum number of days back to go when loading records; if zero or omitted there is no limit (for example, specifying 7 would return all comments in the past week).

The result of **getcomments** is an array of comment records, each of which has the following fields:

| Field | Type | Meaning |
|---------|--------|----------------------------------|
| user | string | Name of user who entered comment |
| userent | string | Enterprise name of user |
| date | date | Date/time of comment |
| subject | string | Comment subject |
| text | string | Comment text |
| emails | string | Comment emails list |

For example:

```
<sf:set n="comms" x="getcomments(item)">
  <sf:if x="length(comms) > 0">
    ... define table for comment display ...

    <sf:for n="ixl" from="0" to="length(comms)-1">
      <sf:set n="c" x="comms[ixl]"/>
    ...
```

Standard Properties

For reports generated from the client, these standard properties are available via the **getprop** function:

pagewidth and **pageheight**: the page size in millimeters¹ as selected by the user.

user: a combined value containing the ID and name of the user generating the report.

userent: a combined value containing the ID and name of the enterprise of the user generating the report.

Configuring XFO Objective Sheet Output

There is a new **ObjectiveSheetType** integrator parameter for the PLX RDO export run type. If the value of this parameter is “default,” then the current objective sheet formatting is used.

Otherwise the general form of the value is:

TYPE[-MODE],prop.a=x,prop.b=y,...

The TYPE is the template type; this should be **xmf** or **xfo**. The optional mode selects a template other than the default. The optional property settings are passed to the template formatter. In an XFO template these may be retrieved using the **getprop** function. This allows a single common template to generate varying output.

Examples:

| Value | Selected file |
|---------------------------|---------------|
| xmf | xmf-19 |
| xfo | xfo-19 |
| xfo-styles | xfo-styles-19 |
| xfo-styles,prop.mode=full | xfo-styles-19 |

The template upload page now includes fields to select the required type and mode.

Configuring XFO Client Printing in Oracle Retail Design

XFO printing is now available in the Oracle Retail Design client as an alternative to the rather basic client-side PDF generation. There are pros and cons for the new approach. While the output will certainly be better, there will be the extra initial effort of template construction. And the XFO output will not reflect exactly what is seen on the screen. It can be thought of as a reporting engine.

XFO printing is the preferred solution for style comment printing.

To configure XFO printing for styles, please reference the Server side reporting definition section in the chapter “General User View Configuration.”

¹ A millimeter is 1/1000 of a meter. See, for example: <http://www.npl.co.uk/reference/length.html>

Spreadsheet Expression Syntax

This document is a brief description of the spreadsheet expression syntax used throughout the configurable definition files of Oracle Retail Design. Expressions can be leveraged in user view configuration files to support the Oracle Retail Design to Oracle Retail Webtrack project integration, in the tab layout definition files to support configuration sheets, and in XFO templates used to support printing and export processes.

Data Types

Values in expressions are either numbers or strings.² A value can also be undefined (represented internally by the special numeric value NaN, or not-a-number). When an undefined value is used in an expression, the result is generally also undefined. There are built-in functions which will test for undefined values.

Numbers are stored in double precision format, with an approximate range of $\pm 5 \cdot 10^{324}$ to $\pm 2 \cdot 10^{308}$. A numeric constant is a sequence of decimal digits, with an optional decimal point and exponent. The exponent part is $e\pm\text{integer}$ (or $E\pm\text{integer}$). If the sign is omitted, a positive exponent is used.

Examples:

1. 1.0 .023 1e4 1E-10 2.3e+5
2. 3e+5 = 2.3x10⁵ = 230000.

A string constant is a sequence of characters enclosed in ' or " quotes. The quote character used to start a string must be used to end it (when entering a string constant in an expression used as an XML attribute, avoid using the quote character used for the attribute).

Within a string the backslash (\) character is used to introduce escape sequences. The following sequences are useful:

| Sequence | Character |
|----------|-----------|
| \\ | \ |
| \" | " |
| \' | ' |

Examples:

"abcd" 'Please type something' "A string with a quote \" inside"

² Values derived from user entry fields can sometimes be *both* a number and a string. For example a country dropdown in Design will be linked to a value that contains the internal ID of the country as a number and the display name of the country as a string. The numeric value is available for parameter lookups, while the string value is used for display in text fields, and so on.

Lists

Value list objects are returned by the **lookup** functions. They are generally used to populate drop down choices in spec sheets and to populate dynamic row and column sets in spec sheet matrices.

The number of items in a list can be determined with the **length** function and individual elements can be obtained by subscription.

Arrays

An array value has any number of dimensions. The number of elements is $b_1 \times b_2 \dots$ where b_i are the bounds of each dimension. The bounds may be obtained using the **length** function and individual elements may be obtained using subscription.

Arrays are used to represent cells in the dynamic row/column areas of matrices in spec sheets. A cell which is either in a dynamic row or column area (but not in both) is represented by a 1-dimensional array, while a cell which is both areas is represented by a 2-dimensional array.

Arrays are also used to represent spec sheet mappings attached to entire forms or matrices, or matrix rows or columns.

When such an array cell is used in an expression, the context of the expression is taken into account to determine which elements of the array are involved. If the cell is used in a dynamic array context with the same “dimensions”³, only a single element will be selected and a normal scalar expression is evaluated.

For example, if `x$3.4` represents a cell in the dynamic area of a matrix, and the expression:

```
x$3.4 * 2
```

is used in a dynamic area in another matrix with matching dimensions, the expression is evaluated once for each element in the array and the result used for the matching element in the destination matrix.

If an array cell is used in a context which has no dimensionality (that is, is not part of a dynamic area in a matrix) or which has different “dimensions”, then the array value as a whole is used. In this case, the only valid use of the cell is for aggregation or subscription.

For example, using the array cell `x$3.4`, the expression:

```
sum(x$3.4)
```

could be used in a numeric field in the spec sheet to display the sum of all the elements in the cell.

Array items

A single-dimensional array may be used directly in an expression by enclosing a list of values in { } brackets.

For example:

```
{ 1, 2, 3, 4, x+y, 'string'}[2]
```

has the value 2. Array items are useful in conjunction with loops in XFO templates.

³ In other words, the “dimension” attribute strings in the dynamic area definitions in the two matrices are equal.

Object Values

An object value is analogous to a Java object with public fields. The value is constructed by client code (often automatically from a real Java object) and made available to the expression evaluation context.

The fields in an object value are accessed using the `->` operator. The right hand side of this operator must be a name.

For example, assuming that a `java.awt.Point` object has been mapped into an object value and stored in 'pt':

```
pt -> x
pt -> y
```

will extract the two fields.

Variable Names

Variables names contain letters, digits, underscore (`_`), dollar (`$`) or period (`.`) characters. A name should start with a letter or digit (names starting with other characters are reserved for internal use).

The special name `$n`, where `n` is a decimal number, represents a function argument. It has no meaning outside of a function definition. `$1` is the first argument; `$2` is the second, and so on. `$0` represents the number of arguments in the call.

Function Calls

A function call is a reference to a built-in or user-defined function. A user defined function can be used to replace commonly-used expressions by a simple call.

The syntax of a call is:

```
functionname(arg1, arg2, ...)
```

There may be zero or more arguments.

Aggregate Functions

Some of the built-in functions operate by *aggregating* the arguments. These functions treat arrays, lists and iterators specially by including all their elements in the aggregation. For example, if `a` and `b` both represent arrays, then `sum(a, b)` will sum all the elements in both arrays.

Expressions

Names, constants and function calls are combined into expressions using *operators*. Operators have differing *precedence*. Higher precedence operators are evaluated before lower precedence operators. Parentheses (()) may be used to alter the order of evaluation.

| Operator | Precedence | Meaning |
|----------|------------|---|
| ?: | 1 | Conditional expression |
| | 2 | OR – the result is 1 if either operand is non-zero, and 0 otherwise |
| & | 3 | AND – the result is 1 if both operands are non-zero and 0 otherwise |
| = or == | 4 | Equals: evaluates to 1 or 0 |
| != or <> | 4 | Not equals |
| < | 5 | Less than |
| <= | 5 | Less than or equals |
| > | 5 | Greater than |
| >= | 5 | Greater than or equals |
| + | 6 | Addition or concatenation |
| - | 6 | Subtraction |
| | 6 | String concatenation |
| * | 7 | Multiplication |
| / | 7 | Division |
| % | 7 | Modulus (remainder) |
| ^ | 8 | Power: $a^b = a^b$ |
| -> | 9 | Field selection |

The addition and comparison operators may be used with string operands; if one operand is a number and the other is a string, the number is converted to a string before evaluation. The addition operator (+) performs string concatenation if either operand is a string; the concatenation operator (||) always converts *both* arguments to strings before evaluation.

For example, 1+2 evaluates to 3, while 1 || 2 evaluates to “1.02.0”.

Other operators evaluate to *undefined* if either argument is a string.

Conditional Expressions

The conditional expression operator ? is used with three operands:

a ? b : c

If a is non-zero the result is b otherwise the result is c.

Subscription

Individual elements of arrays and lists may be obtained using subscripts in [] brackets. Arrays with more than one dimension require multiple subscripts separated by commas. All subscripts are zero-based. An alternative is to use the **element** function.

Iterator Expression

Iterator expressions can be used to perform an aggregate calculation with an expression calculated over all the elements of an array. Iterators are recognized only as the arguments to aggregate functions such as **sum** or **avg**.

The selection of elements in the array is performed by the evaluation client, possibly using constraints to limit the set returned.

The syntax is:

```
{ name : expression }
```

The expression is evaluated over all the elements of the array represented by 'name'. The value is *undefined* if the client does not support iteration over the array.

Iterators are commonly used in spec sheet matrices to perform some complex aggregation over the elements in a dynamic row or column set.

For example, assuming that $x_{3.4}$ represents a matrix cell in a dynamic region:

```
sum({ x3.4 : x3.4 ^ 2 }
```

will sum the squares of all the values in the set.

```
avg({ x3.4 : lookup('parameter', x3.4)
```

will average the results of the lookup call over all the elements in the set.

Examples

| Expression | Notes |
|--|---|
| 1+2 | |
| 1+2*3^4 | This is equivalent to $1 + (2 * (3^4))$ |
| ((1+2)*3)^4 | |
| a ^ 0.5 | The square root of a |
| a = b | If a equals b then evaluate to 1 otherwise evaluate to 0. |
| a = b & c = d | If a equals b and c equals d then evaluate to 1 otherwise evaluate to 0. |
| sum(a,b) < 10 c >= 5 | If sum(a,b) is less than 10 or c is greater than or equal to 5, then evaluate to 1 otherwise 0. |
| val3 = 10 ? a+b : a-abs(zz) | If val3 equals 10, then the result is a+b; otherwise the result is a-abs(zz) |
| x[i] | The i'th element in the array or list x. |
| y[1, e+7, n] | An element in the 3-dimensional array y. |
| value -> name | The field 'name' in the object value represented by 'value' |
| value[i] -> name (value[i]) -> name | The field 'name' in the object value stored in the i'th element of the array 'value'. |
| value -> items[x] (value -> items)[x] | The x'th element in the array stored in the field 'items' in the object value 'value'. |
| avg({ x\$3.4 : abs(x\$3.4) }) | Average the absolute value of the elements in the array represented by x\$3.4. |

Built-in Functions

A number of built-in functions are available for use in expressions. Some are available in all contexts; some are specific to spec sheets in all contexts⁴, and some are specific to Oracle Retail Design. If a function is used with an incorrect number of arguments, or arguments of the wrong type, the result is *undefined*.

Functions Available in All Contexts

| Function | Arguments | Meaning |
|--|------------------------|---|
| number(a) | Number | Return a as a number. This is used for 'combined' values which would otherwise be used as strings in expressions. |
| floor(a) | Number | Return the largest integer which is not greater than a: floor(1.9) = 1 and floor(-1.9) = -2 |
| ceil(a) | Number | Return the smallest integer which is not less than a: ceil(1.1) = 2 and ceil(-1.1) = 1 |
| round(a) | Number | Round a to nearest integer |
| abs(a) | Number | Absolute value of a |
| isset(a) | Any | If a is defined, return 1; otherwise return 0 |
| ifset(a, b) | Any | If a is defined, return a otherwise return b |
| zerop(a, b) | Numbers | If a is zero, return 0 otherwise return a*b; this function is useful because b need not be defined if a is zero. |
| if(a1, b1, a2, b2 ..) | Any | If a1 is non-zero, the result is b1; otherwise if a2 is non-zero, the result is b2, and so on. If none of the conditions succeed, the result is undefined if there is an even number of arguments, or the last argument if there an odd number of arguments. if(a, b, c) is equivalent to (a ? b : c). |
| length(a) [§] or length(arr, index) | Array, list or string. | In the single argument form, return the length of the one dimensional array, list or string a. If a is a string the length is the number of characters in the string. In the two argument form, return the length of the dimension index in the array arr. If arr is not an array the result is undefined. |

⁴ For example, within the Spectrum spec sheet application.

| Function | Arguments | Meaning |
|---|--------------------|--|
| substr(str, m) or substr(str, m, n) | String and numbers | This is equivalent to the Oracle SUBSTR function. The first argument is converted to a string. The result is the substring starting at position m which is n characters long. If n is omitted, the remainder of the string is returned. |
| indexOf(a, b) or indexOf(a, b, c) | Strings | In the two argument form, return the position of the substring b in the string a, or -1 if the substring is not found. In the three-argument form, start the search at position c. This is analogous to a.indexOf(b) or a.indexOf(b, c) in Java. |
| lower(a) | String | Convert the string a to lower case. |
| upper(a) | String | Convert the string a to upper case. |
| element(a, x ₁ , ...) | Array or list | Return the element with subscripts x ₁ , ... from the array or list a. This is exactly equivalent to a[x ₁ , ...] - the [] subscripting syntax was introduced after element. |

Aggregate Functions

| Function | Arguments | Meaning |
|-------------------|--------------------|--|
| sum(a, b, c ...) | Numbers | Sum all the arguments; if any are undefined the result is undefined. |
| zsum(a, b, c ...) | Numbers | Sum all the arguments, ignoring any that are undefined. |
| avg(a, b, c ...) | Numbers | Average of the arguments; if any are undefined the result is undefined. |
| zavg(a, b, c ...) | Numbers | Average of the arguments, ignoring any that are undefined. |
| max(a, b, c ...) | Numbers or strings | Maximum value of all the arguments; if any are strings, the result is a string otherwise the result is a number. |
| min(a, b, c ...) | Numbers or strings | Minimum value of all the arguments. |

Lookup Functions in Spec Sheets

The first argument of each lookup function must be a string defining is the parameter key, as set in the parameters admin window. The argument will often be a string constant, but this is not mandatory.

The second argument of the **lookupdate**, **lookuplistdate** and **lookupnosortdate** functions is a date, from a date entry field on a spec sheet or from a linked date value.

The remaining arguments are the lookup keys for the parameter. If the number of key arguments in a single value lookup does not match the definition of the parameter, the result is undefined.

| Function | Arguments | Meaning |
|--|-----------|--|
| lookup(parm, k1, k2 ...) | Any | Lookup single value using current date |
| lookupdate(parm, date, k1, k2 ...) | Any | Lookup single value using supplied date. |
| lookuplist(parm, k1, k2 ...) | Any | Lookup list using current date. If the keys are omitted, all the active values for the parameter are returned. If the parameter type is string, the list results are sorted alphabetically. |
| lookuplistdate(parm, date, k1, k2 ...) | Any | Lookup list using supplied date. If the keys are omitted, all the active values for the parameter are returned. If the parameter type is string, the list results are sorted alphabetically. |
| lookupnosort(parm, k1, k2 ...) | Any | As lookuplist except that the results are not sorted alphabetically. The results are sorted by key. |
| lookupnosortdate(parm, data, k1, k2 ...) | Any | As lookuplistdate except that the results are not sorted alphabetically. The results are sorted by key. |

In a 'list' lookup call, fewer keys than required by the parameter may be used. The result is the set of values matching the supplied keys. For the 'nosort' functions (new in 11.0), the set is sorted by the remaining keys. For example, if a parameter 'list' is defined with two freeform keys, the call:

```
lookupnosort('list', 'one')
```

will return all the values with 'one' in the first key, sorted by the value in the second key. This allows finer control over the ordering of values than with the 'sort' functions which sort by the values.

Formatting Functions in Spec Sheets

These functions may be used to format numeric and date values.

| Function | Arguments | Meaning |
|---|----------------|--|
| <code>format(number, pattern)</code> | Number, string | Format the number using pattern ⁵ . |
| <code>localdateformat(date)</code> or <code>localdateformat(date, pattern)</code> | Number, string | Format the ‘local’ date using pattern ⁶ ; if the pattern is omitted or the pattern value is a number instead of a string, use a locale specific default date format. A numeric pattern value of 1 selects a default date/time format. |
| <code>dateformat(date)</code> or <code>dateformat(date, pattern)</code> | Number, string | Format the ‘system’ date using pattern; if the pattern is omitted, use a locale specific default date format. |
| <code>dateformattz(date, tz)</code> or <code>dateformattz(date, tz, pattern)</code> | Number, string | Format the date using the pattern and the timezone specified by <code>tz</code> . The timezone should be one of the names recognized by <code>java.util.TimeZone</code> . The pattern is selected as in <code>localdateformat</code> . |
| <code>formatfraction(x, base)</code> | Numbers | Format the number <code>x</code> as an integer + fraction, using a fraction divisor of <code>base - 8</code> or <code>16</code> for example. <code>formatfraction(3.375, 16) = ‘3 3/8’</code> |
| <code>formatfraction4(x)</code> | Number | Format the number <code>x</code> as a fraction using base 4. Fractions will be represented using the characters $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$. |
| <code>dateparse(str, pattern)</code> | Strings | Parse the date string using the pattern and return a ‘system date’. For example: <code>dateparse(‘2004-12-23’, ‘yyyy-MM-dd’)</code> |
| <code>localdateparse(str, pattern)</code> | Strings | Parse the date string using the pattern and return a ‘local’ date. |
| <code>dateparsetz(str, pattern, tz)</code> | Strings | Parse the date string using the pattern and the timezone specified by <code>tz</code> . |

⁵ See the documentation for `java.text.DecimalFormat` for details.

⁶ See the documentation for `java.text.SimpleDateFormat` for details.

Style Linkage Functions in Design

These functions are used to extract list values from styles in the Design application.

| Function | Arguments | Meaning |
|---------------------------------|-----------|--|
| getsizeindex(index) | Number | Get the index'th size name from the current size range. The first name has index 1; getsizeindex(0) returns the number of sizes in the range. |
| getsizelist(sizerange) | Number | Get the sizes in the sizerange as a list, suitable for use in dynamic matrix row or column sets. sizerange must be style\$sizerange or a cell set in a custom size selector component. |
| getsizerratios(sizerange) | Number | Get the ratios in the sizerange as a list |
| getsizecodes(sizerange) | Number | Get the size codes in the sizerange as a list |
| getcolourlist(colset) | String | Get the set of colours as a list, suitable for use in dynamic matrix row or column sets. colset must be style\$colourlist or a cell set in a custom colour selector component. |
| getretailerdata(type [,filter]) | String | Get retailer-specific data identified by type. See below for more information. |
| getsupplierdata(type [,filter]) | String | Get data for the supplier on the style. See below. |
| getagentdata(type[, filter]) | String | Get data for the agent on the style. See below. |

getretailerdata

The first argument to **getretaildata** specifies the data returned. In most cases the result is a value list suitable for use in drop down choices or matrix dynamic row/column sets.

In most cases, the filter, if present, should evaluate to the internal ID of a database value. A 'combined' value resulting from a style linkage or drop down choice is usually used.

For example:

```
getretailerdata('subclasses', style$class)
```

would return a list of all the subclasses in the class used on the style.

| Type | Filter | Meaning |
|--------------|----------|--|
| users | Optional | List of retailer users. If filter is present and non-zero, limit to account managers only. |
| themes | | Themes |
| phases | | Phases in the current season |
| classes | Optional | List of classes. If operating in a single class context, this is just a single value list. If filter is not present, return classes in the current department. Otherwise filter must evaluate to a department ID; the classes in that department are returned. |
| subclasses | Required | List of subclasses. If operating in a single subclass context, this is just a single value list. Otherwise filter must be present and evaluate to a class ID. The subclasses in this class are returned. |
| foblocations | Optional | List of FOB locations. If filter is present it must evaluate to a country ID used to filter the location list. |
| fobleadtime | Required | The integer lead time for the FOB location identified by filter. This is a single value, not a list. |
| fobcountry | Required | The country value for the FOB location identified by filter. This is a single value, not a list. |
| countries | | List of countries. |
| fobcountries | | List of countries which have FOB locations. |
| colours | Optional | List of colours. If filter is present and non-zero, the list is limited to just the colours used in palettes available in the current context. |

The following types are not available only in reporting search sheets; they are not available in spec sheets used with style definition.

| Type | Filter | Meaning |
|-------------|----------|---|
| departments | | List of departments. If operating at department level, this is just a single value list. |
| sizecharts | | List of sizecharts for the context. |
| partners | Optional | List of partners for the current context. If filter is present it must evaluate to a department ID used to filter the partners. |
| status | | List of retailer status values. |
| types | | List of product types |
| elctypes | | List of ELC types |

getsupplierdata and getagentdata

| Type | Filter | Meaning |
|-----------------|----------|--|
| users | Optional | List of partner users. If filter is present and non-zero, limit to account managers only. |
| locations | Optional | List of partner locations (factories, and so on). If filter is present it must evaluate to a country ID used to filter the list. |
| depts | | Partner departments. |
| locationcountry | | The country associated with a partner location (factory). Note that this is a single value, not a list. |

Style Linkages

These are the special names which are linked to style values in Oracle Retail Design. The explanation of each refers to the default field names on the standard summary screen.

In Oracle Retail Design, there are two distinct modes for the summary screen. In ‘standard’ mode, the built-in, hard-coded summary screen is used. In ‘custom’ mode there is no built-in summary screen; all values are set via fields in spec sheets.

- If the Set? column contains No, the value may never be updated from a spec sheet; if the column contains Always, the value may be updated in ‘standard’ or ‘custom’ summary mode; otherwise the value may be updated in ‘custom’ summary mode only.
- If the 11.0? column contains Y, the linkage is available in ‘custom’ summary mode only.
- A type of ‘Combined’ means that the value has both an internal ID, suitable for key lookup, and also a display name.
- A ‘Date’ value is a number, but also contains a formatted date for use in text fields, and so on.

| Link name | Type | Meaning | Set? | 11.0? |
|--------------------|----------|---|-------------------|-------|
| style\$customer | Combined | Retailer | No | |
| style\$stylenumber | String | Style number | | |
| style\$shortname | String | Style short name | | |
| style\$longname | String | Style long name | | Y |
| style\$division | Combined | Division | No | |
| style\$department | Combined | Department | No | |
| style\$season | Combined | Season | | |
| style\$local | Number | 1 if the user belongs to the retailer, 0 if the user belongs to a partner. | No | Y |
| style\$level | Number | Indicates current hierarchy level – 2 for department, 3 for class and 4 for subclass. | No | Y |
| style\$class | Combined | Class | Note ⁷ | |
| style\$subclass | Combined | Sub class | Note ⁸ | |
| style\$createdate | Date | Style creation date | No | Y |
| style\$changedate | Date | Style last change date | No | Y |
| style\$buyer | Combined | Buyer user | | |

⁷ style\$class may not be set if operating at class or subclass level

⁸ style\$subclass may not be set if operating at subclass level.

| Link name | Type | Meaning | Set? | 11.0? |
|--------------------|----------|--------------------------|--------------------|-------|
| style\$quality | Combined | Quality user | | |
| style\$designer | Combined | Designer user | | |
| style\$theme | Combined | Theme | | |
| style\$phase | Combined | Phase | | |
| style\$bidline | Date | Bid deadline | | |
| style\$deliveries | Number | # of deliveries | | |
| style\$iadate | Date | IA date | | |
| style\$elctarget | Number | ELC target | Always | |
| style\$fobloc | Combined | FOB Location | | |
| style\$fobcountry | Combined | FOB Country | | |
| style\$priceby | Number | Price by option | | |
| style\$costby | Number | Cost by option | | |
| style\$quantity | Number | Quantity | | |
| style\$price | Number | Retail price | | |
| style\$vat | Number | VAT | | |
| style\$itemcost | Number | Item cost | Always | |
| style\$elc | Number | ELC | Always | |
| style\$margin | Number | Buying margin | No | |
| style\$type | Combined | Product type | Note ⁹ | |
| style\$elctype | Combined | ELC type | Note ¹⁰ | |
| style\$cust_status | Combined | Customer status | Note ¹¹ | |
| style\$sup_status | Combined | Customer status | Note ¹² | Y |
| style\$leaddays | Number | Lead days | | |
| style\$orderby | Date | Order by date | | |
| style\$supplier | Combined | Style supplier | No | |
| style\$accnum | String | Supplier account number | No | |
| style\$accmgr | Combined | Supplier account manager | No | |
| style\$agent | Combined | Agent | No | |

⁹ Must be set from a custom *type* field.

¹⁰ Must be set from a custom *elctype* field.

¹¹ Must be set from a custom *rstatus* field.

¹² Must be set from a custom *sstatus* field.

| Link name | Type | Meaning | Set? | 11.0? |
|------------------------|----------|----------------------------|--------------------|-------|
| style\$agentcontact | Combined | Agent Contact | No | |
| style\$sup_design_id | String | Supplier Design ID | | |
| style\$sup_designer | Combined | Supplier designer | | |
| style\$sup_team | Combined | Supplier team (department) | | |
| style\$factory | Combined | Factory | | |
| style\$country | Combined | COO | | |
| style\$effective_until | Date | Effective until date | | |
| style\$sup_quantity | Number | Quantity offered | | |
| style\$colourlist | String | Style colour list | Note ¹³ | |
| style\$sizerange | Combined | Style size range | Note ¹⁴ | |
| style\$text1 | String | Features | Always | |
| style\$text2 | String | Comments | Always | |
| style\$text3 | String | Proposed Changes | Always | |
| style\$category | String | Category | | |
| style\$dispatch | String | Dispatch type | | |
| style\$misc_N | Any | Misc value N | Always | |
| style\$link_N | Any | Linkage value N | Always | |

Style Linkage Functions in Design

These functions are used to extract list values from styles in the Oracle Retail Design application.

| Function | Arguments | Meaning |
|------------------------|-----------|--|
| getsizeindex(index) | Number | Get the index'th size name from the current size range. The first name has index 1; getsizeindex(0) returns the number of sizes in the range. |
| getsizelist(sizerange) | Number | Get the sizes in the sizerange as a list, suitable for use in dynamic matrix row or column sets. sizerange must be style\$sizerange or a cell set in a custom size selector component. |
| getcolourlist(colset) | String | Get the set of color as a list, suitable for use in dynamic matrix row or column sets. colset must be style\$colourlist or a cell set in a custom color selector component. |

¹³ Must be set from a custom *colourselector* field.

¹⁴ Must be set from a custom *sizeselector* field.

Style Custom Spec Sheet Fields

Design provides a number of ‘custom’ spec sheet fields which are used to set special values on a style. Details are shown below. All can use the standard **width**, **map**, and **loglabel** attributes, as well as the field layout attributes.

Retailer Status

```
<Custom label="A label" name="rstatus" cell="style$cust_stateid"/>
```

In ‘custom’ summary mode the cell name is ignored and should be omitted.

Supplier Status

```
<Custom label="A label" name="sstatus"/>
```

Colour List

The *colourselector* component is used to select a set of colours from palettes. It will normally be used to set the main colour list for a style (**style\$colourlist**), but additional components can be used to create new colour lists, perhaps to support multiple channels, and so on.

In ‘standard’ summary mode, the cell cannot be **style\$colourlist** because this is set on the summary tab.

```
<Custom name="colourselector" cell="style$colourlist">
  <Parameter name="label">true or false</Parameter>
  <Parameter name="codes">true or false</Parameter>
  <Parameter name="colours">true or false</Parameter>
  <Parameter name="replaceable">true or false</Parameter>
</Custom>
```

| Parameter | Meaning | Default |
|-------------|--|---------|
| label | If true, the standard ‘Colours:’ label is shown above the selector. | false |
| codes | If true, the colour code is shown next to the name. | false |
| colours | If true, the actual colours (using the RGB values) are displayed. | false |
| replaceable | If true, the Replace Colour button is shown. This option has no effect if any of the built-in costing tabs (volumes, bom, price and cost) are present because these do not support colour replacement. | false |

Size Range

The *sizeselector* component is used to select a size range (chart). It will normally be used to set the main size range for a style (**style\$Sizerange**), but additional components can be used to set extra size ranges, perhaps to support multiple channels, and so on.

In 'standard' summary mode, the cell cannot be **style\$Sizerange** because this is set on the summary tab.

```
<Custom name="sizeselector" cell="style$Sizerange">
  <Parameter name="label">true or false</Parameter>
  <Parameter name="codes">true or false</Parameter>
  <Parameter name="colours">true or false</Parameter>
  <Parameter name="replaceable">true or false</Parameter>
</Custom>
```

| Parameter | Meaning | Default |
|-----------|--|---------|
| label | If true, the standard 'Size Chart:' label is shown above the selector. | false |
| chart | If true, the size list is show below the range drop down. | true |
| sizes | If true, size names are shown in the chart. | true |
| ratios | If true, initial size ratios are shown as percentages. | false |
| codes | If true, size codes are displayed in the chart. | false |

Documents

The *documents* component implements the attachment documents feature.

In 11.0, this custom component is not available in 'standard' summary mode and also must be used on a fixed spec tab. It cannot be used on the specification, bid or elc tabs.

In the 11.1 release, additional documents components may be used. There is a single *main* documents area, corresponding to the documents area on the standard fixed summary screen. The main area has the same restrictions on use as in the 11.0 release.

Additional documents areas are identified with an **imageid** attribute. This must be unique within the spec sheet and is used to identify the attached documents in the database. The **imageid** value must end with an asterisk (*) to indicate to the spec sheet subsystem that multiple documents are associated with the component.

```
<Custom name="documents">
  <Parameter name="label">true or false</Parameter>
</Custom>
```

```
<Custom name="documents" imageid="mydocs*">
  <Parameter name="fontsize">3</Parameter>
  <Parameter name="size">300</Parameter>
</Custom>
```

| Parameter | Meaning | Default |
|-----------|---|---------|
| label | If <i>true</i> , the standard ‘Documents:’ label is shown above the component. | false |
| fontsize | Font size increment for names in document list; works in same way as similarly named specsheet item attribute. | 0 |
| size | Sets target size for document list box. If the value is a single integer, it sets the height of the box. Otherwise the value should be WxH, where W and H are integers setting the width and height. Note that the actual size of the box will be affected by the filling attribute of the component. | 100x75 |

Images

The *images* component provides the standard multiple image tabs. This custom component is not available in ‘standard’ summary mode and also must be used on a fixed spec tab. It cannot be used on the specification, bid or elc tabs.

```
<Custom name="images">
  <Parameter name="number">N</Parameter>
</Custom>
```

| Parameter | Meaning | Default |
|-----------|--|---------|
| number | The number of image tabs shown. Must be in range 1-16. | 8 |

Product Type and ELC Type

The *type* and *elctype* component provides drop downs to select the product type and ELC type. These custom components are not available in ‘standard’ summary mode and also must be used on a fixed spec tab – they cannot be used on the specification, bid or elc tabs.

```
<Custom label="Some label" name="type"/>
<Custom label="Some label" name="elctype"/>
```

Comment Display

The *comments* component is used to display style file comments. It can be used as part of the comment entry form to display the current set of comments. There is no input value associated with the component so a cell name should not be present.

```
<Custom name="comments">
  <Parameter name="mode">text, list or both</Parameter>
  <Parameter name="initial">text or list</Parameter>
  <Parameter name="print">>true or false</Parameter>
  <Parameter name="textfontsize">N</Parameter>
  <Parameter name="textsize">WxH</Parameter>
  <Parameter name="listrows">N</Parameter>
  <Parameter name="showent">other, partner, always or never</Parameter>
</Custom>
```

| Parameter | Meaning | Default |
|--------------|---|---------|
| mode | If text, the comments are shown in text format, most recent first; if list, the comments are displayed in multi-column sortable list format. If both, radio buttons are available to switch between the formats. | both |
| initial | If the mode is both, this selects the initial display | text |
| print | Selects whether the separate Print Comments button is available. (Note that if server-side comment reporting is not configured, comments are always printed in text format and the mode must be text or both to enable printing). | true |
| textfontsize | A positive or negative increment which is added to the point size of the default font to get the font for text format displays. For example, if the value is 2, the font will be 2 points larger than the default. | 0 |
| textsize | The dimension in pixels of the text display area. This may be adjusted to change the size of the comments entry dialogue. The actual size may be affected by screen size or by the size of the list format component. | 900x275 |
| listrows | The number of visible rows in the list format display (note that this refers to rows with a single line of text). The actual number of rows displayed may be larger if the comments component is stretched to fit a form or if the text format is made larger. | 15 |
| showent | Controls whether the enterprise of the comment user is shown in brackets after the user name: other: the enterprise is shown if it is not the same as that of the current user. partner: the enterprise is shown if the user belongs to a trading partner. always: the enterprise is always shown. never: the enterprise is never shown. The value of this parameter is passed to the server template as the showent property. | other |

Appendix: designconfig.dtd

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- DTD for Design configuration definition

    A single file can contain any number of configurations, but these are
    treated independently.

    Common ParameterGroup elements can be included at the top level and
    referred to using ref= attributes.
-->

<!ELEMENT Configurations ((Configuration | ParameterGroup)+)>

<!-- An individual Configuration can be attached to a number of different
scopes.  It defines the tab layout for design windows in each scope.

    The required name attribute supplies a name for the configuration for use
    in GUIs, etc.  The name must be unique amongst other configurations.  When
    a configuration is uploaded, all existing instances with the same name
    are removed and replaced with the new version.  If the upload does not
    contain the same scope set as the database, old scope configurations will
    be lost.

    When a configuration with a new name is uploaded, existing configurations
    in the same scope(s) are not removed.  The Design client uses the most
    recent upload for the required scope; when a style is created or edited,
    the configuration name is saved to the database and used if the style is
    exported.

    When there are multiple configurations and scopes in the database, the
    client locates a configuration using these rules:

    1. Find best hierarchy match
    2. If there is more than match, choose configuration with
       matching season
    3. If there is still more than one match, choose most recently
       uploaded version.

    The hierarchy match is based on the selections on the user console.  If
    the enterprise is using subscoping at the subclass level, but the user
    does not select a subclass, then the match will be by division,
    department and class only.
-->

<!ELEMENT Configuration (Scopes?, Details)>

<!ATTLIST Configuration name      CDATA #REQUIRED
                        version CDATA #IMPLIED>

<!-- The Scopes element contains the individual scopes that the configuration
applies to.  If Scopes is omitted the configuration is the global
default.
-->

<!ELEMENT Scopes (Scope+)>

<!-- A Scope element defines a configuration scope by hierarchy and season.
```

The hierarchy can be defined at division, department, class or subclass level. Design always used the most specific match.

Note that configuration scopes at class and subclass level are relevant only for enterprises using the new 'userscopelevel' option, and for users who select a subscope on the design console. If a user does not select a subscope level then the default configuration for the department or division will be used.

Note that any number of seasons may be included; the configuration is stored for each.

A Scope element without any nested hierarchy or season settings will also be treated as the global default.

```
-->
<!ELEMENT Scope ((Division, (Department, (Class, Subclass?)?)?)?, Season*)>
<!-- Individual scope elements. At least one of Name or Number is always
      required.
-->

<!ELEMENT Division (Name?, Number?)>
<!ELEMENT Department (Name?, Number?)>
<!ELEMENT Class (Name?, Number?)>
<!ELEMENT Subclass (Name?, Number?)>
<!ELEMENT Season (Name?, Number?)>

<!ELEMENT Name (#PCDATA)>
<!ELEMENT Number (#PCDATA)>

<!-- Configuration details. The Tabs element defines the tab layout and the
      optional Conditions element defines the data required before a style can
      be saved.

      If the Tabs element is omitted, the default tab set is used.

      Parameter elements may be used to define configuration parameters. These
      will affect global functions such as style copy.
-->

<!ELEMENT Details (Tabs?, Conditions?, (Parameter | ParameterGroup)*, Exporter?)>
<!-- Design tabs. These can be selected from a set of built-in tabs, fixed
      spec tabs and custom tabs define by classname.

      If the standard Summary tab is used it must be the first in the list.

      A specific builtin tab may not be listed more than once. Each tab may
      include Parameter elements to provide further configuration information.
      Currently Summary and Builtin tabs will ignore parameters.
-->

<!ELEMENT Tabs (Summary?, (Builtin | Spec | Custom)*)>

<!ELEMENT Summary ((Parameter | ParameterGroup)*)>

<!-- The type attribute for Builtin defines the standard tab type -->

<!ELEMENT Builtin (Parameter*)>
<!ATTLIST Builtin type (volumes | price | bom | cost | specification | bid | elc |
labels) #REQUIRED>
```

<!-- The Spec tab defines a fixed spec sheet. The type attribute specifies the numeric sheet type; the specsheet 'application code' is fixed. The key attribute is used to locate saved data for the tab; it must be unique within the configuration.

The perm attribute controls whether the tab is visible to the current user. The value is treated in the same way as the perm attribute in spec sheets, which controls edit access (for details see specsheet.dtd). Note that the tab is always created internally, to ensure that linkage calculations, etc, are performed correctly; the perm value controls whether the tab is made visible.

The TabLabel element defines the label attached to the tab. This may be expanded to allow multiple languages later.

-->

```
<!ELEMENT Spec (TabLabel, (Parameter | ParameterGroup)*)>
<!ATTLIST Spec type CDATA #REQUIRED
              key CDATA #REQUIRED
              perm CDATA #IMPLIED>
```

```
<!ELEMENT TabLabel (#PCDATA)>
```

<!-- A Custom tab is defined by a Java class name. -->

```
<!ELEMENT Custom (TabLabel, (Parameter | ParameterGroup)*)>
<!ATTLIST Custom class CDATA #REQUIRED
                perm CDATA #IMPLIED>
```

<!-- Parameter defines a single extra configuration parameter for a tab. -->

```
<!ELEMENT Parameter (#PCDATA)>
<!ATTLIST Parameter name CDATA #REQUIRED>
```

<!-- ParameterGroup allows a set of related parameters to be grouped together and accessed using the group name. It can be used to set parameters for a single area of the application, such as reporting.

Note that although the syntax allows ParameterGroups wherever Parameters are valid, the implementation code will often not use groups.

A common ParameterGroup can be defined with an id attribute in a configuration or at the top level and referred to using the ref attribute. If ref= is used, the name attribute must not be present and there must be no nested Parameter elements. The id name must be unique amongst all id attributes in the file.

-->

```
<!ELEMENT ParameterGroup ((Parameter | ParameterGroup)*)>
<!ATTLIST ParameterGroup id ID #IMPLIED
                          ref IDREF #IMPLIED
                          name CDATA #IMPLIED>
```

<!-- The Conditions element defines fields which must be set before a style can be saved.

-->

```
<!ELEMENT Conditions (Condition+)>
```

<!-- A single Condition include a field name attribute and an optional error message. If the error message is omitted, the 'Save' button will not be enabled if the field is not set.

If the standard Summary tab is present, these fields are limited to the special 'linkage values' - misc_X and link_X; otherwise any standard link field can be tested.

In non-summary mode, include the fields 'shortname' and 'stylenumber' to mimic the original behaviour.

Note that the field names are the same as the standard style specsheet values without the leading "style\$".

-->

```
<!ELEMENT Condition (Error?)>
<!ATTLIST Condition field CDATA #REQUIRED>
<!ELEMENT Error      (#PCDATA)>
```

<!-- The Exporter element defines a styletab spec sheet type and linkage information required for style exports.

There is no stored data associated with the export definition form. All information is communicated via linkage cells.

The optional Conditions element defines the names that must be set before any exports are enabled.

Parameter elements can be used to define export preprocessors and other data.

-->

```
<!ELEMENT Exporter (Conditions?, (Parameter | ParameterGroup)*, Export+)>
<!ATTLIST Exporter type CDATA #REQUIRED>
```

<!-- The Export element defines a single export type that is supported by the GUI.

The key attribute identifies the export type in the style XML extract.

The flag attribute defines the characters used to identify the export in the style record. All the export flags in use are combined into one column in the database, so keep the flag strings short.

If the flag is omitted, the style will not be marked for direct export; however the preprocessing will still be performed. This can be used to implement a 'get new style numbers from somewhere else' interface.

The indicator attribute is the cell which is set by the GUI to indicate that the export type has been selected. It should start with the export prefix "exp\$".

The miscdate attribute is the number of a style misc field which is set to the timestamp that the export was requested. It can be used to display export info in the styles list.

If the onceonly attribute is true, the export type should not be enabled if any of the selected styles have already been exported.

-->

```
<!ELEMENT Export (Conditions?, ExportData*)>
<!ATTLIST Export key      CDATA      #REQUIRED
                 flag     CDATA      #IMPLIED
                 indicator CDATA      #REQUIRED
                 miscdate  CDATA      #IMPLIED
                 onceonly  (false | true) "false">
```

<!-- The ExportData element defines additional values which can be set for an

export. The values are output in the styles XML extract.

The key attribute is displayed in the extract; the link attribute defines the export name from the GUI; as with the indicator, it must start with "exp\$". The type attribute controls the display format of the value in the extract.

-->

```
<!ELEMENT ExportData EMPTY>
<!ATTLIST ExportData key CDATA #REQUIRED
                    link CDATA #REQUIRED
                    type (string | int | float | date | time) "string">
```

Appendix: specsheet.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
```

<!-- DTD for Design spec sheets. Version 0.01 -->

<!-- SpecSheets is the enclosing element. It contains one or more individual spec sheet definitions.

The optional application attribute defines the the internal "application code" for the sheets. The sheet processor can reject an import if it is configured for a different code.

The version attribute can define a default version for all the sheets in the file. The default is 2.

```
-->
```

<!ELEMENT SpecSheets (SpecSheet+)>

<!ATTLIST SpecSheets application CDATA #IMPLIED
version (1 | 2) "2">

<!-- SpecSheet defines a single sheet layout. It contains one or more pages.

The integer type attribute defines the specification type number.

The description attribute defines text displayed for the sheet in administration tools.

If the tabbed attribute is present tabs are displayed; if defaulttabs is present the (very old) default tab titles are used. This attribute is deprecated and will be removed eventually.

The fill attribute defines whether the sheet fills horizontally. The default is false.

The version attribute defines the spec sheet display version. Currently versions 1 and 2 are supported. The default is the SpecSheets version attribute.

The background attribute sets the background colour for the page and its forms. It should be a colour name like 'black' (see the fields of the java.awt.Color class), or an RGB value in decimal or hex (decoded by java.lang.Integer.decode).

If the background attribute is not present, the background colour is 'light gray' (the standard Design colour) for version 1 and white for version 2 sheets.

The borders attribute sets the default border for all forms in the sheet; if omitted the default is 'etched' for version 1 and 'lined' for version 2.

The enabled attribute defines an expression which can be used to dynamically update the 'editing enabled' state of the sheet. The attribute is available on most nested elements; the most recently seen value will override the settings on enclosing elements.

Note that enabled settings do not combine. A form may have a 'true' enabled attribute whilst the value for the sheet is false; the form will

then be available for editing.

The perm attribute defines editing permissions for the specsheet. The basic format of the perm value is:

allow-deny

'allow' and 'deny' are comma-separated lists of application-specific security roles. The roles are the keys defined in the system administration services window (the "sec:" prefix may be omitted)

"*" means 'everyone'; it should not be used in conjunction with any other roles. An empty list means 'nobody'.

The deny part may be omitted if there are no exclusions.

Examples:

perm="*" means 'everybody' and has the same effect as omitting the attribute.

perm="loc-1" means all users in the 'loc' group except those in the local '1' group. It is equivalent to perm="sec:loc-sec:1".

perm="*-2,p1" means everybody, except those users in the local '2' group and the partner '1' group.

Common settings will be perm="loc" for local users only and perm="prt" for partner users only.

Not all combinations are sensible: for example "loc-*" means 'all local users except for everybody', and is the same as perm="". An item which cannot be edited by anyone is not very useful.

The perm attribute is available on most other nested elements; the most recently seen value will override settings on enclosing elements.

-->

```
<!ELEMENT SpecSheet (Page+)>
<!ATTLIST SpecSheet type          CDATA          #REQUIRED
                    description CDATA          #REQUIRED
                    tabbed       (true | false)   "false"
                    defaulttabs (true | false)   "false"
                    fill         (true | false)   "false"
                    version      (1 | 2)         #IMPLIED
                    borders      (none | lined | etched) #IMPLIED
                    background   CDATA          #IMPLIED
                    enabled      CDATA          #IMPLIED
                    perm         CDATA          #IMPLIED>
```

<!-- The Page element is a single page or tab.

If the sheet is not tabbed there must be one page only. Each Page contains one or more matrices or forms elements.

The title attribute defines the tab title. It is required if the sheet is tabbed.

The scrollable attribute defines whether the entire page scrolls. The default is true.

The background attribute sets the background colour of the page; if

```

    omitted the default for the sheet is used.
-->

<!ELEMENT Page ((Matrix | Form)*)>
<!ATTLIST Page title      CDATA          #IMPLIED
                scrollable (true | false) "true"
                background CDATA          #IMPLIED
                enabled    CDATA          #IMPLIED
                perm       CDATA          #IMPLIED>

```

<!-- The Matrix element defines a multi-column list.

The name attribute is the unique name for this matrix in the sheet. The name is always required. The name is not defined as an XML ID because the same name may be used in different SpecSheets.

The description attribute is used to generate a heading for the matrix.

The rows attribute defines the number of rows; the number of columns is implied by the number of Column elements.

The headings attribute defines whether headings are displayed.

The visiblerows attribute defines the number of rows displayed in the client GUI. The default is 5. The actual number of rows displayed may differ if the matrix size is altered to fit in the available space.

If the scrollable attribute is absent, the matrix scrolls vertically if the Page is not scrollable, and does not scroll if the Page is scrollable. This default behaviour is suitable for fixed matrices, but may not be best for matrices with dynamic row sets.

The horizontalscroll attribute specifies whether a horizontal scroll bar is present in the Matrix. This can be useful if there is a dynamic column set. If horizontal scrolling is selected, the visiblecols attribute gives an indication as to the visible width of the matrix. The measure is based on a notional default column width and does not relate directly to real columns in the matrix.

The leftfixedcols is the number of non-scrolling columns on the left of the matrix. It can be used only if horizontalscroll is true and must be less than the number of columns. A ColumnSet counts as a single column in the leftfixedcols count.

The rowheadingwidth attribute must be present if row headings are required; it is the width of the headings column. If a dynamic row set with subrows (see below) is present, there will be two headings columns; rowheadingwidth may be a pair of integers separated by a comma in this case. If a single integer is given it is used for both columns.

The showeditable attribute can be false, true or a colour name (see background attribute for details of colour names). If not false, editable cells will be displayed with a different background colour in edit mode.

The visible attribute defines an expression which can be used to dynamically show or hide the matrix. The Matrix will be visible if the expression evaluates to a non-zero number.

The cellpfx attribute must be set if any cells in the matrix are involved in spreadsheet calculations. It is used as a prefix to form the spreadsheet cellnames for matrix cells. A cell at row 'r' and column 'c' will have a spreadsheet cellname of pfx\$r.c. For example, if the prefix

is "zz", the cell at row 3, column 6 will have a spreadsheet cellname of zz\$3.5. If no cells are involved in calculations then omitting the cellpfx attribute will speed up several matrix operations.

The map attribute defines a mapping for the entire matrix.

Initial contents and row/cell mappings are defined by Row elements.

Calc elements may be included before Column elements to define intermediate values and functions.

The optional CellChoice element defines the default items for any choice columns or cells in the matrix. It can be included even if the default cell type is not choice; this allows the default to be used for a number of individual choice cells without repeating the items.

CalcSet elements are used to compute 1-d and 2-d array values in dynamic row and/or column areas.

-->

```
<!ELEMENT Matrix ((Calc | CalcSet)*, CellChoice?, (Column | ColumnSet)+, (Row | RowSet)*)>
```

```
<!ATTLIST Matrix name          CDATA          #REQUIRED
                  description   CDATA          #IMPLIED
                  border        (none | lined | etched) #IMPLIED
                  rows          CDATA          #REQUIRED
                  headings      (false | true)   "true"
                  visiblerows   CDATA          #IMPLIED
                  visiblecols   CDATA          #IMPLIED
                  scrollable     (false | true)   #IMPLIED
                  horizontalscroll (false | true)  "false"
                  leftfixedcols CDATA          #IMPLIED
                  rowheadingwidth CDATA          #IMPLIED
                  showeditable  CDATA          #IMPLIED
                  visible       CDATA          #IMPLIED
                  cellpfx       CDATA          #IMPLIED
                  map           CDATA          #IMPLIED>
```

<!-- Entity defining cell attributes. These attributes can be set in Matrix, Column or Cell elements. The Cell setting will override the Column setting which in turn overrides the Matrix setting.

The align attribute defines the column alignment. For example, 'l' would be used for text items, 'r' for numeric values and 'c' for date values. The alignment can be overridden by an align attribute in a Cell element.

The type attribute defines the datatype expected for values in the column.

The prec attribute can be used with float only; it defines the number of decimal places; the default is 2.

The limit attribute can be used with text cells only; it defines the maximum text length. It must be 'none' (to cancel an enclosing default limit) or a positive integer.

-->

```
<!ENTITY % cellattrs 'align (l | c | r)
#IMPLIED
                        type (text | int | float | date | checkbox | choice)
#IMPLIED
                        prec CDATA
#IMPLIED
```

```

        limit    CDATA
#IMPLIED
        enabled CDATA
#IMPLIED
        perm    CDATA
#IMPLIED'>

<!ATTLIST Matrix %cellattrs;>

<!-- The column attribute defines a single column. The width is required, but
the heading is optional. If the heading is omitted but the matrix
headings attribute is true, an empty heading is displayed.

The column heading may be set in a heading attribute or child Heading
element, but not both. If the headingexpr attribute is true, the heading
value is treated as an expression which is evaluated to obtain the column
heading.

The map attribute defines a column level mapping.

The optional CellChoice element defines the default items for choice
cells in the column.
-->

<!ELEMENT Column (Heading?, CellChoice?)>
<!ATTLIST Column width    CDATA          #REQUIRED
                 heading  CDATA          #IMPLIED
                 headingexpr (false | true) "false"
                 map      CDATA          #IMPLIED
                 %cellattrs;>

<!-- A ColumnSet attribute defines a dynamic 'column' set. This contains a
set attribute defining an expression which should evaluate to a list of
column headings. A ColumnSet is treated as a single 'virtual' column when
referred to in spreadsheet expressions and Cell elements. The
spreadsheet value associated with a cell in a dynamic column set will be
a 1-d or 2-d array.

The dimension attribute defines the data; it can be used in spreadsheet
calculations to link with related data in other matrices. For example,
the dimension could be "size" for a dynamic column set derived from a
list of sizes. If the same dimension is used elsewhere in a row or
column set, the two sets can be used in linked calculations.

If the matrix has a cellpfx attribute, and a dimension is provided, a
spreadsheet cell named "PFX$DIMENSION" will be set to the list of values
from the set expression. Here PFX is the cell prefix and DIMENSION is
the dimension. For example, if the cellpfx is "szmat" then the cell
containing the size names in the example above would be szmat$size.

If the required attribute is true, the matrix will not be displayed if
the set is empty.

Currently the expression used to define the set should not contain direct
or indirect references to other cells in this matrix.
-->

<!ELEMENT ColumnSet (CellChoice?, SubColumn*)>
<!ATTLIST ColumnSet width    CDATA          #IMPLIED
                    set      CDATA          #REQUIRED
                    dimension CDATA          #IMPLIED
                    required (false | true) "false"
                    map      CDATA          #IMPLIED

```

```

                                %cellattrs;>

<!-- A SubColumn element defines a sub column within a dynamic column
set. Cell level attributes may be specified in the ColumnSet or SubColumn
elements; the latter always takes precedence. A map attribute may not be
used in ColumnSet if there are SubColumns. A width must be specified in
the ColumnSet or SubColumn.
-->

<!ELEMENT SubColumn (CellChoice?)>
<!-- ATTLLIST SubColumn width          CDATA          #IMPLIED
                                heading       CDATA          #REQUIRED
                                map           CDATA          #IMPLIED
                                %cellattrs;>

<!-- The Heading element defines the heading for a matrix column -->

<!ELEMENT Heading (#PCDATA)>

<!-- The Row element defines initial contents for a matrix row. The row
attribute is the one-based row number. The heading attribute is the row
heading; it will be ignored if the matrix rowheadingwidth attribute is
not present. If headingexpr is true, the heading is treated as an
expression.

The map attribute defines a row-level mapping.
-->

<!ELEMENT Row (Cell*)>
<!-- ATTLLIST Row row                CDATA          #IMPLIED
                                heading       CDATA          #IMPLIED
                                headingexpr   (false | true) "false"
                                map           CDATA          #IMPLIED>

<!-- A dynamic row 'set' may be defined with the RowSet element. This
contains a set attribute defining an expression which should evaluate to
a list of row headings. A dynamic row set can also contain nested
'sub-row' SubRow elements. The subrows are numbered in the same sequence
as normal rows. For example:

    <RowSet row="5" set="someexpr">
      <SubRow heading="sub1"/>
      <SubRow heading="sub2"/>
    </RowSet>

This defines rows 5 and 6. Nested sub-rows are allowed with RowSets
only.

The spreadsheet cell associated with a Cell in a RowSet will be an array;
two-dimensional if the Cell is part of a dynamic column set.

See the notes on ColumnSet for a description of the dimension and
required attributes.

A cell will be defined containing the row set list if the matrix has a
cellpfx and the dimension is specified - for more details see the
ColumnSet notes.

A map attribute is not allowed if the RowSet contains sub-rows.

Currently the expression used to define the set should contain direct or
indirect references to other cells in this matrix.
-->

```

```
<!ELEMENT RowSet (SubRow* | Cell*)>
<!ATTLIST RowSet row      CDATA      #IMPLIED
                  set      CDATA      #REQUIRED
                  dimension CDATA      #IMPLIED
                  required (false | true) "false"
                  map      CDATA      #IMPLIED>
```

```
<!ELEMENT SubRow (Cell*)>
<!ATTLIST SubRow heading CDATA #IMPLIED
                  map      CDATA #IMPLIED>
```

<!-- The cell element defines initial contents of a row/column cell and can also define a cell mapping. The col attribute is the 1-based column number. This attribute allows only the non-default Cell elements to be included.

The contents of the Cell are the initial value; the editable attribute defines whether the cell can be edited by the user. If editable is false, and there is no initial value, the cell will be blank.

If the heading attribute is true the cell contents are displayed differently, perhaps using a bold font.

The background attribute sets the background colour of the cell. This should be used only with a cell that has editable false or has a non-overridable expression (otherwise the standard background/editable colours are used).

The expr attribute defines a spreadsheet expression which is used to determine the cell contents. Any cell with an expression is implicitly non-editable.

Note that an editable cell which has initial contents must be used with care. Once any edits have been made to the matrix in the client, the initial value will be saved to the server. The saved value will then continue to be used, even if the initial value is changed in the matrix definition.

If the cell type is 'choice', a CellChoice element may be present to define the choice items. There must be a CellChoice defined in the Cell, Column or Matrix. Note that DTD rules require that the contents are surrounded by (*) indicating any number of CellChoice elements. However it is not legal to include more than one.

The initial value for a checkbox cell must be true or false; false is the default.

If there is an initial value, it should be defined by a single Value element; for compatibility with previous versions, the value can be defined directly in the Cell if there are no other elements. If there is a CellChoice element, the initial value is not allowed.

loglabel and loglabelx define the change log label for the cell. loglabel is a simple string, loglabelx is an expression evaluated in the context of the cell. loglabel and loglabelx cannot both be used.

-->

```
<!ELEMENT Cell (#PCDATA | CellChoice | Value)*>
<!ATTLIST Cell col      CDATA      #REQUIRED
                  editable (true | false) "true"
                  heading (true | false) "false"
                  background CDATA      #IMPLIED>
```

```

        expr      CDATA      #IMPLIED
        loglabel  CDATA      #IMPLIED
        loglabelx CDATA      #IMPLIED
        map       CDATA      #IMPLIED
    %cellattrs;>

<!-- The CellChoice element defines the contents of a matrix 'choice' cell.
CellChoice is similar to the basic Choice item, but has far fewer
attributes. For clarity in this definition, and ease of parsing, a
separate element is used.

    Note that all CellChoice elements are keyed.
-->

<!ELEMENT CellChoice (Option+)>
<!ATTLIST CellChoice keyed (true | false) #FIXED "true">

<!-- CalcSet defines array calculations in a dynamic matrix.

    The dimensions attribute lists the dimension values used to construct the
    array. It must contain one or two dimension strings, separated by
    commas, which match the dimensions of the ColumnSet or RowSet elements.

    If the flexible attribute is true, the expression is evaluated with
    flexible dimension matching - this allows array values to be shared
    between matrices or spreadsheets when the dimensions do not match
    exactly.
-->

<!ELEMENT CalcSet EMPTY>
<!ATTLIST CalcSet cell      CDATA      #REQUIRED
                  expr      CDATA      #REQUIRED
                  dimensions CDATA      #REQUIRED
                  flexible   (false | true) "false"
                  map        CDATA      #IMPLIED>

<!-- The Form element defines a spec sheet 'form' with input items.

    The mandatory name attribute defines the form name. The name is used to
    match the form against input data stored in the database.

    The description attribute is used to generate a heading for the form.

    The columns attribute defines the number of columns used in the display.
    The default is 1.

    The tabbed attribute is true if the form items are displayed as separate
    tabs.

    The tabsinrow attribute is relevant only if the form is tabbed; it
    defines the number of per row.

    The vertical attribute selects an 'old-style' vertical set of textareas
    format.

    Note that we allow direct nesting Matrix and Form elements without requiring
    a SubForm. A SubForm is needed only if attributes are required.

    The visible attribute defines an expression which can be used to
    dynamically show or hide the form. The form will be visible if the
    expression evaluates to a non-zero number.

    The background attribute sets the background colour; if omitted the

```

colour is taken from the parent form or page.

The border attribute sets the border for the form.

Form level mapping is defined by the map attribute.

-->

```
<!ELEMENT Form (Defaults | TextField | IntField | FloatField | TextArea | Choice |
Image | Icon | Checkbox |
SubForm | Label | MultiLabel | DateField | Custom | Matrix | Form
| Calc)*>
```

```
<!ATTLIST Form name          CDATA          #REQUIRED
description CDATA          #IMPLIED
columns      CDATA          "1"
tabbed       (false | true)  "false"
tabsinrow    CDATA          #IMPLIED
vertical     (false | true)  "false"
scrollable   (false | true)  "false"
map          CDATA          #IMPLIED
enabled      CDATA          #IMPLIED
visible      CDATA          #IMPLIED
background   CDATA          #IMPLIED
border       (none | lined | etched) #IMPLIED
perm         CDATA          #IMPLIED>
```

<!-- Entity defining basic attributes for all items

The label attribute defines the label attached to the item. For 'vertical' forms it is the heading; for tabbed forms it is the tab name. For normal components, the labelfont and labelfontsize attributes define the font for the label. The labelalign attribute defines the label alignment. If the 'labelcolon' attribute is true (the default), a colon is appended to the label string.

The cellw and cellh attributes define the number of form cells that the item occupies, vertically and horizontally. The defaults are 1. cellw must not be greater than the number of columns in the form.

The labelw attribute defines the number of 'half-cells' that are allocated to the label. It must be in the range 1 to (cellw*2-1). The default is (cellw*2-1) which means that if cellw is more than one, the label occupies all but one of the half-cells.

The fill attribute defines whether the item expands to fill the available space. Possible values are 'n' for no filling, 'h' for horizontal filling, 'v' for vertical filling and 'b' for filling in both dimensions. The default is no filling, except for text (and int/float) fields for which the default is 'h' to facilitate easy alignment of fields.

The pos attribute defines the position the item occupies if it is smaller than the available space. Possible values are 'c' for centre and the compass points. The default depends on the item type.

The font attribute is 'b' for a bold font, 'i' for italic and 'bi' for bold italic.

The fontsize attribute defines the increment over the default font. Positive values are larger than the default, negative values smaller.

The loglabel attribute defines the text that appears in the change log for the field; if omitted the label is used. A loglabel should be provided if the label is omitted for a field which should be present in

```

    the log.
-->

<!ENTITY % itemattrs 'label          CDATA
#IMPLIED
                        width          CDATA          "0"
                        cellw         CDATA
#IMPLIED
                        cellh         CDATA
#IMPLIED
                        fill          (n | h | v | b)
#IMPLIED
                        pos           (c | n | ne | e | se | s | sw | w | nw)
#IMPLIED
                        font          (b | i | bi)
#IMPLIED
                        fontsize     CDATA
#IMPLIED
                        labelw        CDATA
#IMPLIED
                        labelfont     (b | i | bi)
#IMPLIED
                        labelfontsize CDATA
#IMPLIED
                        labelalign    (l | c | r)
#IMPLIED
                        labelcolon    (false | true)      "true"
                        loglabel     CDATA
#IMPLIED
                        perm          CDATA
#IMPLIED'>

<!-- Attributes for items which can also have a map attribute -->

<!ENTITY % mapattrs '%itemattrs;
                    map CDATA #IMPLIED'>

<!-- Attributes for items which can have a cell attribute.

The index attribute is the 1-based index of the item in the associated
data array.  If the attribute is omitted the first free index is used.

An index value must not be used more than once.  Setting an index
attribute allows the item to be moved around in form without disturbing
saved values.

Various arrays in the client, server and database are sized according to
the maximum index used in a form.  It is important that these values are
kept as small as possible.  No index value (explicit or implicit) more
than 1024 is allowed.

The form attribute is the name of a form which stores the data for the
field.  The name must refer to a Form element, not a Matrix.  The index
attribute must be present if form is used.  Care must be taken when
allocating indices in forms which are used elsewhere - a reference to
index in a form from an earlier form will 'reserve' that index.
-->

<!ENTITY % itemcellattrs '%mapattrs;
                        index CDATA #IMPLIED
                        form  CDATA #IMPLIED
                        enabled CDATA #IMPLIED
                        cell  CDATA #IMPLIED'>

```

```

<!-- Attributes for items which can have expr and cell attributes -->

<!ENTITY % itemexprattrs '%itemcellattrs;
                        expr CDATA #IMPLIED'>

<!-- The compat attribute is used with elements which do not have any
associated data and which were supported in the old 'forms' language.
These items were counted as part of the data store for the form, even
though there could be no data. If an old form has existing data stored
in the database, this attribute should be set to true to ensure that the
column positions are preserved.

The compat attribute is not required with new elements such as Defaults
or Calc.
-->

<!ENTITY % datalessattrs '%itemattrs;
                        compat (true | false) "false"'>

<!-- The Defaults element sets defaults for the basic item attributes -->

<!ELEMENT Defaults EMPTY>
<!ATTLIST Defaults %itemattrs;>

<!-- Common 'initial value' element. With some elements a value attribute can
be used as an alternative. The value element and attribute must not be
used together.
-->

<!ELEMENT Value (#PCDATA)>

<!-- Simple item types. Some can have the default value as an attribute -->

<!-- Simple text fields. The optional limit attribute (a positive integer)
sets the maximum text length.
-->

<!ELEMENT TextField (Value?)>
<!ATTLIST TextField %itemexprattrs;
                limit CDATA #IMPLIED
                value CDATA #IMPLIED>

<!ELEMENT IntField (Value?)>
<!ATTLIST IntField %itemexprattrs;
                value CDATA #IMPLIED>

<!-- Floating point number.

The prec attribute defines the number of decimal places; the default is 2.
-->

<!ELEMENT FloatField (Value?)>
<!ATTLIST FloatField %itemexprattrs;
                value CDATA #IMPLIED
                prec CDATA #IMPLIED>

<!-- The rows attribute for a text area is the number of text rows -->

<!ELEMENT TextArea (Value?)>
<!ATTLIST TextArea %itemexprattrs;
                rows CDATA #REQUIRED>

```

```

<!-- Fixed image from icon list.

The iconname attribute defines the name of the icon. The dim attribute
defines the display size of the image; if omitted the actual image size
is used.

dim must be WxH where W and H are integers.
-->

<!ELEMENT Icon EMPTY>
<!ATTLIST Icon %datalessattrs;
            iconname CDATA #REQUIRED
            dim      CDATA #IMPLIED>

<!-- Checkbox items have a 'state' attribute to set the initial value.

A checkbox with a group attribute acts like a 'radio' button - only one
checkbox in the group is set at a time.
-->

<!ELEMENT Checkbox EMPTY>
<!ATTLIST Checkbox %itemcellattrs;
            state (false | true) "false"
            group CDATA          #IMPLIED>

<!ELEMENT Label (Value?)>
<!ATTLIST Label %datalessattrs;
            align (l | r | c) #IMPLIED
            value CDATA       #IMPLIED>

<!-- Multi-line wrapped label.

The dim attribute defines the display width in pixels. The default is 300.
-->

<!ELEMENT MultiLabel (Value?)>
<!ATTLIST MultiLabel %datalessattrs;
            dim CDATA #IMPLIED>

<!-- Date entry field. The mode attribute defines the operation and value
returned. It must be one of:

std      date defined using 'standard' server timezone
local    date defined using local timezone
time     date & time defined using local timezone.

The default is 'std'. Note that the default implied by the old forms
language was 'local'.

The icon attribute controls whether the 'set date' button is displayed
with an icon or text. The default is currently 'false'.

A DateField can be used to display the value of an expression.
-->

<!ELEMENT DateField (Value?)>
<!ATTLIST DateField %itemexprattrs;
            value CDATA          #IMPLIED
            mode (std | local | time) "std"
            icon (true | false)  #IMPLIED>

<!-- Custom item require a name attribute to define the item. It can have
nested parameter elements to provide further information to the component

```

creator.

If one or more images can be associated with the item, the imageid attribute is required. If the item can store multiple images, the id must be suffixed with an asterisk (*). The imageid value must be unique within the sheet.

-->

```
<!ELEMENT Custom (Parameter*, Value?)>
<!ATTLIST Custom %itemexprattrs;
            name CDATA #REQUIRED
            imageid CDATA #IMPLIED>
```

<!-- Parameter defines a single extra configuration parameter for a custom component -->

```
<!ELEMENT Parameter (#PCDATA)>
<!ATTLIST Parameter name CDATA #REQUIRED>
```

<!-- The subform item encloses a nested form or matrix.

If the nested item is omitted, a name must be present referring to an earlier form. These shared forms must not contain, directly or indirectly, any data entry items.

-->

```
<!ELEMENT SubForm ((Matrix | Form)?)>
<!ATTLIST SubForm %datalessattrs;
            name CDATA #IMPLIED>
```

<!-- The Choice item contains a number of options. Each option element contains the displayed choice and an optional 'value' which is used for external export and updates. The first item is the default, unless an option is present with the 'selected' attribute set to true.

By default the value stored in the database for a Choice item is the index of the selected option. This means that if options are added or the list is reordered, the stored value may refer to a different option. To avoid this, a Choice item can be 'keyed'. In this case each option must include a distinct 'key' attribute. The key is stored in the database, allowing option lists to be changed without disturbing the value.

-->

```
<!ELEMENT Choice (Option+)>
<!ATTLIST Choice %itemcellattrs;
            keyed (true | false) "false">
```

<!-- Items in a Choice. The key attribute is compulsory if the Choice is keyed.

The list attribute defines a lookup parameter code which is used to get the real options for this item. All the active (and current) values for the parameter are included. If there is a default value for the parameter, it is included as the first item; the remaining items are alpha sorted. If list is used, the option contents must be empty.

If a list option is marked as the default (with selected = true), then the default item is the first in the list.

The expr attribute defines a spreadsheet expression which evaluates to the real option(s) for this item. expr is similar to list except that the set of value(s) can depend on other values in the spreadsheet.

The value attribute defined the string used for import and export processes to refer to the option. If value is not present, the string itself is used. The value attribute is ignored if a value list is set.

-->

```
<!ELEMENT Option (#PCDATA)>
<!ATTLIST Option value CDATA #IMPLIED
                 selected (true | false) "false"
                 key CDATA #IMPLIED
                 list CDATA #IMPLIED
                 expr CDATA #IMPLIED>
```

<!-- The Image item contains either a single name or a set of named options.

The dim attribute defines the display size of the image. If omitted fixed defaults are used.

dim must be WxH where W and H are integers.

If compact is 'true', the is shown without the notes, change and fullsize buttons; instead a pop-up menu is available for these functions.

-->

```
<!ELEMENT Image (ImageOption+)>
<!ATTLIST Image %datalessattrs;
             dim CDATA #IMPLIED
             compact (true | false) "false">
```

<!-- The name attribute in an ImageOption must be unique within the form -->

```
<!ELEMENT ImageOption (#PCDATA)>
<!ATTLIST ImageOption name CDATA #REQUIRED>
```

<!-- The Calc element defines a spreadsheet cell with associated expression. Both cell name and expression are required attributes.

If the function attribute is true, the element defines a spreadsheet function; the cell name is the function name and the expression is the body.

The map attribute defines a mapping for the calculated value. map cannot be used with function declarations.

The Calc element is not associated with any form of GUI component.

-->

```
<!ELEMENT Calc EMPTY>
<!ATTLIST Calc cell CDATA #REQUIRED
               expr CDATA #REQUIRED
               function (true | false) "false"
               map CDATA #IMPLIED>
```