Sun Desktop Manager 1.0 Developer Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–2728
January, 2006

060223@13996

Contents

Preface

The *Sun Desktop Manager 1.0 Developer Guide* provides guidelines for developers who want to enable applications for the Sun™ Desktop Manager 1.0 Beta. It provides the necessary knowledge about how to centrally manage the configuration of software applications that are not recognized by the Desktop Manager by default.

After you have read this document, you will be able to create and deploy files, called "templates", that contain information about where to store and how to display new configuration settings. This document also provides design recommendations, information about creating advanced templates, and reference information, which will help you to build the templates that you need.

Who Should Use This Book

The *Sun Desktop Manager 1.0 Developer Guide* is aimed at developers and advanced site administrators who want to extend the Sun Desktop Manager to be able to centrally configure additional applications and settings.

Before You Read This Book

It is recommended that you read at least the "Concepts and Architecture" chapter in the *Sun Desktop Manager 1.0 Administration Guide*, and have some experience in administering and using the Desktop Manager. Some knowledge of XML is helpful, but not essential.

How This Book Is Organized

Chapter 1 provides an overview of the Sun Desktop Manager 1.0.

Chapter 2 provides an introduction to templates and how to create them.

Chapter 3 describes how to create and use more complex templates.

Chapter 4 discusses guidelines for design recommendations.

Chapter 5 provides information about configuration concepts.

Appendix A provides helpful information about configuration path mapping.

Appendix B provides a reference for template elements and attributes.

Appendix C contains the template DTD.

Related Books

The following Sun documents are related to this manual and can provide you with additional information:

- *Sun Desktop Manager 1.0 Administration Guide*
- *Sun Desktop Manager 1.0 Installation Guide*

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`

◆ ◆ ◆ CHAPTER 1

Sun™ Desktop Manager 1.0 Overview

This chapter provides an overview of the Sun Desktop Manager 1.0 Beta and an introduction to the concepts needed to create templates for the Desktop Manager.

To introduce a new application to the Desktop Manager, you need to develop templates. You also need to register those templates with the Desktop Manager. *Templates* are files that contain information about where to store and how to display new configuration settings. The Desktop Manager uses these templates to obtain all necessary information about configuration profiles.

Overview

The Desktop Manager provides the necessary infrastructure for a centralized configuration of the Sun Desktop Manager. Currently, the Desktop Manager consists of the following client- and server-side components:

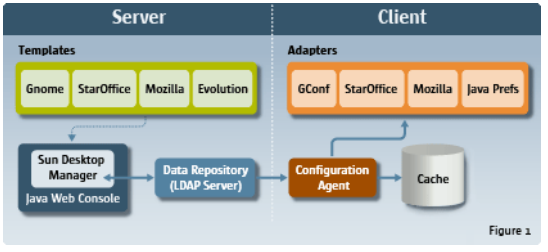


FIGURE 1–1 Client- and server-side components

Configuration Propagation

All policies are stored in a central configuration repository, such as an LDAP server. A *profile* is the term for a group of semantically coherent configuration settings. A Configuration Agent, running on each client machine, is responsible for retrieving the profile data from the LDAP server, and for

caching the data locally. The Configuration Agent periodically checks for any changes on the LDAP server, and updates the cache accordingly. Furthermore, the Configuration Agent sends notifications to all interested applications. Desktop applications, such as StarOffice, Mozilla, Evolution or GNOME, read the policies by means of corresponding adapters. These adapters encapsulate the necessary communication with the cache and the Configuration Agent.

Configuration Management

The Desktop Manager is a web-based administration tool that allows you to view, define, and enforce configuration settings on different levels of an organization's hierarchy, such as an organization, group or user level with a web browser. The Desktop Manager is a part of the Java Web Console, which provides the necessary infrastructure for all of Sun's administration tools, such as a common web-based graphical user interface (GUI) and single sign-on authentication. The Desktop Manager uses *templates* to view, define, and enforce configuration settings in the configuration repository and to render the GUI for displaying these configuration settings.

◆ ◆ ◆ CHAPTER 2

Templates

Desktop Manager templates provide information about the location of every configuration setting in the configuration repository. Templates also provide information about their visual representation in the GUI of the Desktop Manager. Templates are XML files that conform to a document type definition (DTD) file.

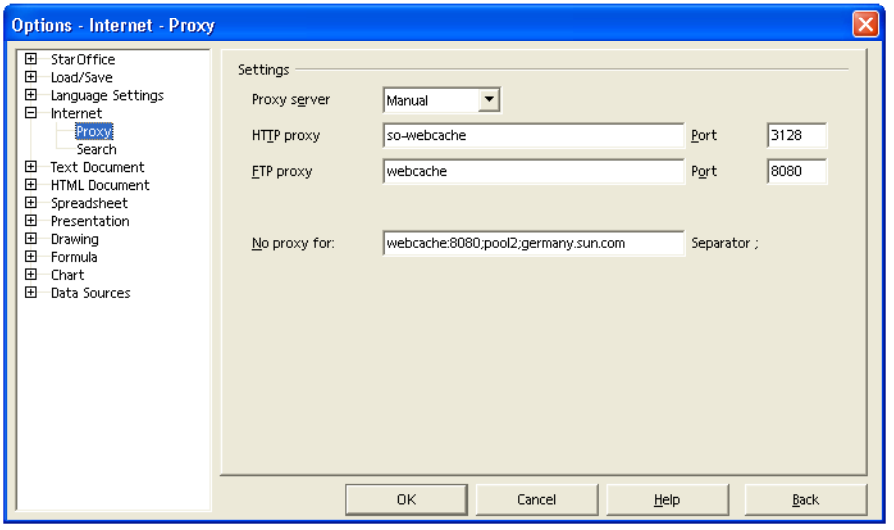
Using XML allows you to create definitions for managing configurations settings that are independent of the GUI rendering engine, the operating system, and the programming language. The GUI is rendered based on the semantic dependencies of the elements specified in the template. Due to its generality, the Desktop Manager template format does not provide solutions for every possible GUI design request. For instance, exact positioning on the screen is not supported.

This chapter contains information about the typical development cycle for templates. Beginning with a currently existing configuration dialog of a desktop application, you will learn how to create a simple template for that dialog. You will also learn how to make that file available to the Desktop Manager, so that the file is displayed in the Content Area.

The “Hello world!” Template

▼ Creating the “Hello world!” template

Before You Begin Assume that you want to make the proxy configuration settings of StarOffice available to the Desktop Manager.



The following template provides a first implementation of the GUI:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE apt:template SYSTEM "policytemplate.dtd">
<apt:template>
  <category apt:name="StarOffice" apt:label="StarOffice">
    <category apt:name="Internet" apt:label="Internet">
      <page apt:name="Proxy" apt:label="Proxy">
        <section apt:name="Settings" apt:label="Settings">
          <property apt:name="ProxyServer" apt:label="Proxy Server"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetProxyType"
                    oor:type="xs:int">
            <visual apt:type="radioButtons"/>
            <constraints>
              <enumeration oor:value="0" apt:label="None"/>
              <enumeration oor:value="2" apt:label="Manual"/>
            </constraints>
          </property>
          <property apt:name="HTTPProxy" apt:label="HTTP Proxy"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetHTTPProxyName"
                    oor:type="xs:string"/>
          <property apt:name="HTTPPort" apt:label="HTTP Port"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetHTTPProxyPort"
                    oor:type="xs:int"/>
          <property apt:name="FTPProxy" apt:label="FTP Proxy"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetFTPProxyName"
                    oor:type="xs:string"/>
          <property apt:name="FTPPort" apt:label="FTP Port"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetFTPProxyPort"
```

```
                    oor:type="xs:int"/>
          <property apt:name="NoProxyFor" apt:label="No Proxy For"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetNoProxy"
                    oor:type="xs:string"/>
        </section>
      </page>
    </category>
  </category>
</apt:template>
```

The following steps are necessary to announce the new template to the Desktop Manager:

1. **Login as `root` to the machine on which you installed the Desktop Manager.**
2. **Create a directory that is called `HelloWorld/templates/StarOffice/Internet/Proxy` under `/usr/share/webconsole/apoc/packages`.**
3. **Create a file called `proxy.xml` with the XML template content that was listed previously. Copy the file to the `Proxy` directory.**
4. **Grant the user "noaccess" read/execute permission to the `Proxy` directory.**
5. **Grant the user "noaccess" read access to the `proxy.xml` file.**
6. **Execute `/usr/sbin/smreg add -a /usr/share/webconsole/apoc`.**
7. **Restart the webserver with the `/usr/sbin/smcwebserver restart` command.**

   After you log in to the Desktop Manager, you should see a new top-level category that is called "StarOffice". Browsing down that category displays the "Proxy" page that you defined with the template that you created.

Explaining the "Hello world!" Template

The first two lines of the template are initial XML definitions. The third line contains the root element of the template called `apt:template`, which encloses the whole definition of the profile made in a template file.

The next four lines contain the main template structure elements. By nesting `apt:category` elements, you create the nodes of the configuration profile tree. The configuration profile tree represents the visual hierarchy of the profiles in the GUI of the Desktop Manager (see "Trees" on page 30). If you specify an `apt:name` attribute, the attribute is used to uniquely designate that element. The `apt:label` attribute specifies the displayed text as a category in the GUI. If you do not specify the `apt:label` attribute, the displayed text is defined by the `apt:name` attribute. Therefore, always specify an `apt:label` element, because this element is used for localization. See "Localization" on page 14.

Every `apt:category` element must contain one or more `apt:category` elements or an `apt:page` element. The `apt:page` element represents a leaf in the configuration profile tree. The “Proxy” page that was shown previously is an example of a leaf. The Desktop Manager renders a page as a single HTML page that has to be divided in at least one `apt:section`. An `apt:section` element renders all its child elements in a table with a table heading. Using more than one section enables you to group settings on one page.

The `apt:section` element contains `apt:property` elements, which represent the configuration settings. The "Hello, world!" template contains six properties: `ProxyServer`, `HTTPProxy`, `HTTPPort`, `FTPProxy`, `FTPPort`, and `NoProxyFor`. Every property contains an `apt:dataPath` attribute. This attribute is required and specifies the data location in the configuration tree. The configuration tree in turn represents the hierarchy of the configuration settings as it is stored in the configuration repository. See “Trees” on page 30 for more information.

The `oor:type` attribute defines the data type of the configuration setting in the configuration repository. `ProxyServer`, `HTTPPort`, and `FTPPort` are of type `xs:int`, the other properties are of type `xs:string`. Integer and string types are displayed as edit fields by default.

The `visual` element is used to instruct the Desktop Manager how to display the property. Without specifying this element, the property `ProxyServer` would have been rendered using an edit field instead of a radio button group. This element is optional.

Tip – The GUI of the Desktop Manager deviates from the original StarOffice GUI by rendering the two possible integer values as a radio button group instead of using a drop-down list. For visualizing a dual value, improved usability is achieved using radio buttons instead of a drop-down list. An example, for instance, is one click compared with two clicks for changing a value.

The `constraints` element, in combination with the `enumeration` sub-element, is used to specify the number of radio buttons rendered and the integer values stored in the back end, depending on which radio button is selected. The `apt:label` attribute specifies the string rendered on the GUI for every radio button.

Localization

It is important to localize all strings that are defined in a template. Localized strings are retrieved from resource files. The `resImport` sub-element of the `apt:template` element is used to bind one or more resource files to a template. You need to specify the fully qualified path to the resource and its base name, for example:

```
<apt:template>
<resImport
  apt:packagePath="com.sun.star.apoc.policies.resource.staroffice"/>
```

The resource key to be used is defined by providing the name of the key as value of the `apt:label` attribute, e.g.

```
<property apt:name="HTTPProxy" apt:label="SO.internet.proxy.name"
          apt:dataPath="org.openoffice.Inet/Settings/ooInetHTTPProxyName"
          oor:type="xs:string">
```

The Desktop Manager searches every resource file bound to a template for the key specified in the `apt:label` attribute first. If no key is found, the value of the `apt:label` attribute is displayed. If a key is found, the corresponding value is retrieved from the resource file and is displayed.

The resource file from which the string is retrieved is determined in a similar way to the mechanism defined for Java: the package path specified in the `resImport` element and the languages selected in the web browser determine the selected resource file. For example, if en_US is the language for web pages selected in the browser, and the package path specified in the `resImport` element is `com.sun.star.apoc.policies.resource.staroffice`, the Desktop Manager searches the following files in the given order for the resource key:

```
./res/com/sun/star/apoc/policies/resource/staroffice_en_US.properties

./res/com/sun/star/apoc/policies/resource/staroffice_en.properties

./res/com/sun/star/apoc/policies/resource/staroffice.properties
```

The Desktop Manager searches the files in the local profile package first (see "The Profile Package Format" on page 15). If they are not found, all other packages are searched. This enables strings that are already localized in other packages to be reused, especially for category names.

See the API specification of the Java ResourceBundle for more details on resource recovery.

Tip – All applications of the Java Web Console determine the language during the login. To force any application to use a new language, you must log out. Then log in again after you have changed the language for web pages in your browser.

The online help should also be localized. The Desktop Manager chooses the HTML file according to the same rules that are applied to the resource files, except that only the local profile package is searched for the help file. For example, if you specify `/StarOffice/Internet/Proxy` as the path to the HTML file, and en_US in your browser, the Desktop Manager displays the online help file `./web/StarOffice/Internet/Proxy_en_US.html`.

The Profile Package Format

Templates are embedded into a deployment container, which is similar to a "package" in the Java™ programming language. A package can also contain optional files, such as resource files for GUI localization, HTML files for online help, and arbitrary support files.

The Desktop Manager uses a special directory and a file name format to access the templates and all necessary optional files. This directory and file name structure is called the *profile package format*.

All profile packages are located in a unique subdirectory below the `/usr/share/webconsole/apoc/packages` directory. For the "Hello, world!" example, `HelloWorld` was chosen, resulting in a `/usr/share/webconsole/apoc/packages/HelloWorld` directory.

Tip – Use the product name and product version of the software for which you are installing the package. This ensures a unique package directory. For instance, `HelloWorld3.1` is a better choice than `HelloWorld`.

Below the specific package directory, which should not be confused with the `packages` directory, the following subdirectories are allowed:

templates	The `templates` subdirectory must contain all templates of the profile package. Files that have the postfix `.xml` are considered to be templates. The name of the file must correlate with the value of the `apt:name` attribute of the `page` element. Templates can be organized in any way, although they should be located in the same directory hierarchy, as specified by their category hierarchy.
classes	The `classes` subdirectory must contain all class files of the profile package. Files that have the postfix `.class` are considered to be Java class files. The name of the file has to correlate with the name of the class defined in that file. The files have to be located in the same directory hierarchy as specified by their package hierarchy.
web	The `web` subdirectory must contain all HTML help files of the profile package and must contain images referenced by the profile package. Files that have the postfix `.html` are considered to be HTML files. Correlate the name of the HTML file with the name of the template using the HTML file. Locate HTML files in the same directory hierarchy as the template using the HTML file.
res	The `res` subdirectory must contain all resource files of the profile package. Files that have the postfix `.properties` are considered to be Java-compliant resource files. You can correlate the names and paths of the resource file with the names and paths of the template files that use them. You can also specify one directory hierarchy containing one resource file for all templates.
lib	The `lib` subdirectory must contain all library files of the profile package. Files that have the postfix `.jar` are considered to be libraries. Libraries are automatically loaded by the Desktop Manager's class loader. Their content is accessed by using the root directory in the jar file as the root directory for absolute paths. The typical use for library files is to act as a container for class or resource files , and for directories that are normally located in the `classes` and `res` directories.

Other file types have no special meaning to the Desktop Manager. Nevertheless, the files types can be placed in the `classes` or `web` directories, if they are needed by class or HTML files.

To further illustrate the package format, Figure 2–1 shows a mature `HelloWorld` package:

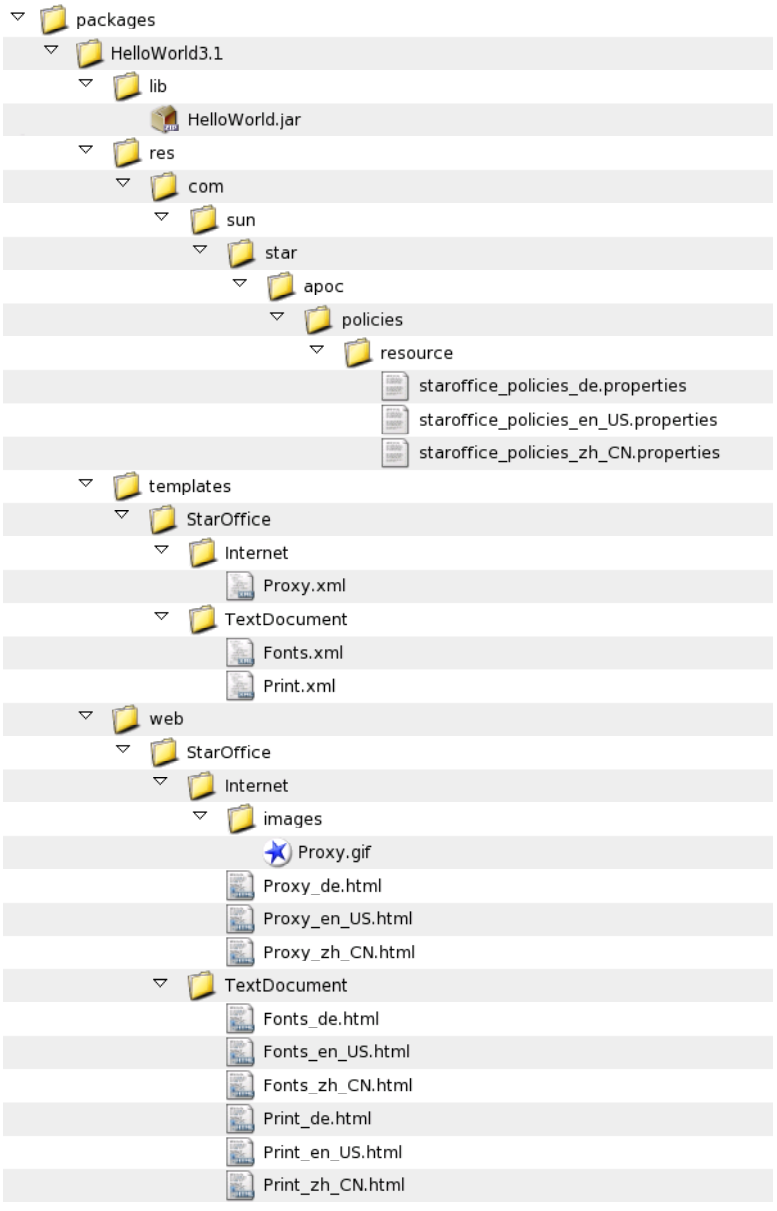


FIGURE 2–1 `HelloWorld` Package

The deployment of packages is completely up to the developer of the package, as long as you follow

the rules defined previously in this chapter. You can provide a collection of files with instructions about how to copy the files to the correct locations, or you can provide a zip file, or you can use the deployment mechanisms provided by the operating systems, such as `.pkg` files for Solaris or `.rpm` files for Linux. The last method is recommended, as the enhanced means for maintaining and removing software offered by the corresponding operating system provide better support for the end user.

◆ ◆ ◆ CHAPTER 3

Advanced Templates

The chapter provides information about building and using more complex templates.

Sets

After creating the "Hello, World!" template presented in Chapter 2, you might need to make the list of configurable proxies dynamic. A dynamic list enables users to add proxies for additional protocols like GOPHER, SOCKS, and SSL . *Sets* enable you to accomplish this task.

Note – The use of sets for the proxy page is for demonstration purposes only. Neither StarOffice nor OpenOffice can handle this example, because their implementation expects the original layout of the configuration tree.

Proxies are added if you click on the New button. A dialog prompts for the name of the new protocol to use a proxy for. Specify **FTP** the first time, then click New again and type **HTTP** as the name for the second protocol. . The two entries that you see are links. If you click on one of the links, the Content Area is loaded with the content.

The functionality is implemented by modifying the "Hello, world!" example as follows: Delete the four properties `HTTPProxy`, `HTTPPort`, `FTPProxy`, and `FTPPort`. Then add a set element, as seen in the annotated section in the following code example:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE apt:template SYSTEM "policytemplate.dtd">
<apt:template>
  <category apt:name="StarOffice" apt:label="StarOffice">
    <category apt:name="Internet" apt:label="Internet">
      <page apt:name="Proxy" apt:label="Proxy">
        <section apt:name="Settings" apt:label="Proxy Server">
          <property apt:name="ProxyServer" apt:label="Proxy Server"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetProxyType"
```

```
                    oor:type="xs:int">
              <visual apt:type="radioButtons"/>
              <constraints>
                <enumeration oor:value="0" apt:label="None"/>
                <enumeration oor:value="2" apt:label="Manual"/>
              </constraints>
            </property>
            <property apt:name="NoProxyFor" apt:label="No Proxy For"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetNoProxy"
                    oor:type="xs:string"/>
          </section>
      <!-- Beginning of set element to be added -->
          <set apt:name="ProxyList" apt:label="Proxy List"
               apt:dataPath="org.openoffice.Inet/Settings/ooInetProxyList">
            <page apt:name="ProxyPage" apt:label="Proxy">
              <section apt:name="Proxy" apt:label="Host and Port">
                <property apt:name="HostName" apt:label="Host Name"
                          apt:dataPath="./$queriedId/HostName"
                          oor:type="xs:string"/>
                <property apt:name="Port" apt:label="Port"
                          apt:dataPath="./$queriedId/Port"
                          oor:type="xs:string"/>
              </section>
            </page>
          </set>
      <!-- End of added set element -->
        </page>
      </category>
    </category>
</apt:template>
```

The `apt:dataPath` attribute of the `set` element points to the place where the set is stored in the back end. The `set` element contains a `page` element, which contains a `section` element, which in turn contains a `property` element. This hierarchy correlates to the element hierarchy below a category element. It is rendered as a page in the same way, except that it is triggered by clicking the link in the set table.

In comparison to a category page, the set page properties `HostName` and `Port` use a special notation for the `apt:dataPath`. The path begins with a dot, meaning that the path is relative to the first path definition found ascending the element hierarchy. The first parent element with anapt:dataPath is the set element, so the Desktop Manager translates the relative path of, for example, the `Port` property to `org.openoffice.Inet/Settings/ooInetProxyList/$queriedId/Port`. The other peculiarity in this path is the `$queriedId` variable. Since all data in the configuration repository needs to be uniquely identified, every element in a dynamic data structure needs to have a unique name. The `$queriedId` variable instructs the Desktop Manager to query the user for that name when the Add button is clicked. The resulting set element is stored with the specified name at the position determined by the position of the variable. Therefore, in the case of the `FTP` set element, the path to its port property is `org.openoffice.Inet/Settings/ooInetProxyList/FTP/Port`.

Another example: You can also use sets to display the `NoProxyFor` property. If the string of host names becomes long, the use of an edit field for this property can be problematic. Users then need to scroll to view the entire string in the edit field. A list of proxy names that are implemented with a set avoids that extra scrolling.

To use a set instead of a string for the `NoProxyFor` property, delete the `NoProxyFor` element with all its sub-elements. Then add the set element that is shown in annotated section in the following code example:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE apt:template SYSTEM "policytemplate.dtd">
<apt:template>
  <category apt:name="StarOffice" apt:label="StarOffice">
    <category apt:name="Internet" apt:label="Internet">
      <page apt:name="Proxy" apt:label="Proxy">
        <section apt:name="Settings" apt:label="Proxy Server">
          <property apt:name="ProxyServer" apt:label="Proxy Server"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetProxyType""
                    oor:type="xs:int">
            <visual apt:type="radioButtons"/>
            <constraints>
              <enumeration oor:value="0" apt:label="None"/>
              <enumeration oor:value="2" apt:label="Manual"/>
            </constraints>
          </property>
        </section>
    <!-- Beginning of set element to be added -->
        <set apt:name="NoProxyFor" apt:label="No Proxy For"
             apt:dataPath="org.openoffice.Inet/Settings/ooInetNoProxySet">
          <page apt:name="HostNamePage">
            <section apt:name="HostNameSection">
              <property apt:name="HostNameProp"
                        apt:dataPath="./$queriedId/HostName" oor:type="xs:string"
                        apt:storeDefault="true">
                <visual apt:type="hidden"/>
                <value>$queriedId</value>
              </property>
            </section>
          </page>
        </set>
    <!-- End of set element to be added -->
        <set apt:name="ProxyList" apt:label="Proxy List"
             apt:dataPath="org.openoffice.Inet/Settings/ooInetProxyList">
          <page apt:name="ProxyPage" apt:label="Proxy">
            <section apt:name="Proxy" apt:label="Host and Port">
              <property apt:name="HostName" apt:label="Host Name"
                        apt:dataPath="./$queriedId/HostName"
                        oor:type="xs:string"/>
              <property apt:name="Port" apt:label="Port"
```

```
                        apt:dataPath="./$queriedId/Port"
                        oor:type="xs:string"/>
            </section>
          </page>
        </set>
      </page>
    </category>
  </category>
</apt:template>
```

The entries in the table are no longer links, but rather the entries represent a flat list of single-valued elements. This is achieved by using the `apt:storeDefault` attribute and the `visual` element in combination with the `value` element. The `value` element can define a default value for a configuration setting. A default value is not stored in the configuration repository by default. The `apt:storeDefault` attribute instructs the Desktop Manager to override that default and to automatically store the default value in the back end. In this case, the default value is the value that the user enters in the dialog when a new set element is added. If the `apt:type` attribute of the `visual` element is specified as "hidden", the only section in that page is left empty. If a set page is empty, it makes no sense to display the page, so the Desktop Manager does not provide a link.

Action Handlers

An action handler is used to execute user-defined actions whenever an event occurs. At the point, only one action handler is available: the XML handler, which generates JavaScript code in the client-side browser.

The XML handler can be used to implement a feature of the StarOffice/ OpenOffice Proxy dialog not yet covered by the templates presented so far: selecting the value "None" for the "Proxy server" setting disables the edit fields.

The annotated areas in the following template show the necessary changes in the original "Hello, world!" template in order to enable or disable the edit fields if the "Proxy server" setting is set to "Manual" or "None":

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE apt:template SYSTEM "policytemplate.dtd">
<apt:template>
  <category apt:name="StarOffice" apt:label="StarOffice" >
    <category apt:name="Internet" apt:label="Internet">
      <page apt:name="Proxy" apt:label="Proxy">
        <section apt:name="Settings" apt:label="Settings">
          <property apt:name="ProxyServer" apt:label="Proxy Server"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetProxyType"
                    oor:type="xs:int"
               <!-- The following line should be added to original "Hello, world!" template -->
                    apt:xmlHandler="switchState">
            <visual apt:type="radioButtons"/>
```

```
            <constraints>
              <enumeration oor:value="0" apt:label="None"/>
              <enumeration oor:value="2" apt:label="Manual"/>
            </constraints>
          </property>
          <property apt:name="HTTPProxy" apt:label="HTTP Proxy"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetHTTPProxyName"
                    oor:type="xs:string"/>
          <property apt:name="HTTPPort" apt:label="HTTP Port"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetHTTPProxyPort"
                    oor:type="xs:int"/>
          <property apt:name="FTPProxy" apt:label="FTP Proxy"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetFTPProxyName"
                    oor:type="xs:string"/>
          <property apt:name="FTPPort" apt:label="FTP Port"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetFTPProxyPort"
                    oor:type="xs:int"/>
          <property apt:name="NoProxyFor" apt:label="No Proxy For"
                    apt:dataPath="org.openoffice.Inet/Settings/ooInetNoProxy"
                    oor:type="xs:string"/>
        </section>
    <!-- Beginning of section to be added to original "Hello, world!" template -->
        <xmlHandler apt:name="switchState">
          <event apt:type="onChange" />
          <action>
            <choose>
              <when apt:test="ProxyServer.value=0">
                <command>HTTPProxy.enabled=false</command>
                <command>HTTPPort.enabled=false</command>
                <command>FTPProxy.enabled=false</command>
                <command>FTPPort.enabled=false</command>
                <command>NoProxyFor.enabled=false</command>
              </when>
              <otherwise>
                <command>HTTPProxy.enabled=true</command>
                <command>HTTPPort.enabled=true</command>
                <command>FTPProxy.enabled=true</command>
                <command>FTPPort.enabled=true</command>
                <command>NoProxyFor.enabled=true</command>
              </otherwise>
            </choose>
          </action>
        </xmlHandler>
    <!-- End of section to be added -->
      </page>
    </category>
  </category>
</apt:template>
```

By adding the `apt:xmlHandler` attribute to the property `ProxyServer`, you associate the `xmlHandler` element with the same name (here: "switchState") to that property.

Action handlers are triggered by events. Which events are considered by an action handler is defined by the `apt:type` attribute of the event element. At this point, there is only one event available: the `onChange` event. This event is thrown when a user enters new data for a property. In the preceding example, the event is used to trigger the XML handler when the value of the `ProxyServer` property changes.

The action element contains the actions that are executed if any of the events specified by the event element occur. In the preceding example, the first action is to check the value of the `ProxyServer` property and to change the state of the other edit fields accordingly. This is achieved by using the `choose`, `when` and `otherwise` elements. All edit fields are disabled, if the `ProxyServer` property is set to "None". The edit fields are enabled, if the `ProxyServer` property is set to "Manual".

Help

There are two different types of help: the online help and the inline help.

The online help is a detailed, context-sensitive help that is displayed in a new window when the user clicks the Help link in the masthead. If the last action was taken in the Content Area, the online help document defined in the template currently displayed in the Content Area is shown. If the last action was taken somewhere else, the general online help is shown. The `apt:onlineHelp` attribute of the `page` element is used to bind the HTML help file to a template. You need to specify the fully qualified path to the file and its base name, for example,

```
<page apt:name="Proxy">
      apt:label="Proxy"
          <apt:onlineHelp="/StarOffice/Internet/proxy">
```

references `./web/StarOffice/Internet/proxy.html`.

You can use absolute and relative paths in HTML files. If you place an image in a directory that is called `images`, alongside of the HTML file, any of the following notations can be used in the HTML file:

- `<img src="/StarOffice/Internet/images/image.gif">`
- `<img src="./images/image.gif">`

The inline help is a brief text providing additional information for category, page and property pages. The inline help is displayed in the "Comment" column for a `category` element, below the page title for a `page` element and below the configuration setting for a `property` element. The help text is specified by the `apt:inlineHelp` attribute of any of the three elements `category`, `page` or `property`. For example,

```
<property apt:name="HTTPProxy"
          apt:label="HTTP Proxy"
          apt:inlineHelp="Specify no proxy (None) or a manually defined proxy (manual)."
```

```
          apt:dataPath="org.openoffice.Inet/Settings/ooInetHTTPProxyName"
          oor:type="xs:string"
</property>
```

You should provide resource keys for the label to facilitate localization.

◆ ◆ ◆ CHAPTER 4

Design Recommendations

Before you create templates for your application, you need to determine which configuration settings are to be centrally managed by the Desktop Manager. The most simple solution is to create templates containing every possible configuration setting, but this approach may create an unnecessary amount of work and often results in settings being displayed by the Desktop Manager that are never used.

Guidelines

Which settings to choose while creating templates depends heavily on the application in question, but the following list may provide some guidelines which settings to use:

- Settings that deal with security.
- Settings that deal with lockdown.
- Settings that reference resources, such as hosts, ports, URLs, paths, and so on.
- Settings that are used to define corporate identity, such as fonts or colors

The concrete names of the categories and pages are completely up the template developer. There are only two rules to obey:

- The names must be unique for every level in the tree so that a unique path can be created, and
- The first category must uniquely identify the application and its version, for example, “StarOffice 6.0” .

Choosing an `apt:section` element or an `apt:page` element should depend on the following two requirements: You should try to avoid scrolling in the Content Area by restricting the amount of configuration items per page to no more than six. If you are mapping an already existing GUI to the Desktop Manager GUI, you should try to map the application GUI to the Desktop Manager GUI as precisely as possible to optimize usability by recognition. If these two requirements contradict, choose the latter one. If you are considering a deviation in the Desktop Manager GUI from an application GUI, it should be a small one.

Every text displayed on the GUI should present a consistent look. Apply the following capitalization guidelines for text that appears in GUI design elements:

- Use sentence capitalization for the inline help and the online help. Capitalize only the first word of each sentence, unless the text contains proper nouns, abbreviations, or acronyms that are always capitalized.
- Observe proper punctuation within and at the end of full sentences.
- Avoid the use of long phrases that are not full sentences. If you must use a phrase that is not a full sentence, no punctuation is required at the end.
- Use headline capitalization for labels, titles, checkbox text, menu and list items. To apply headline capitalization, capitalize every word except articles ("a", "an", and "the"), coordinating conjunctions (for example, "and", "or", "but", "so", "yet", and "nor"), and prepositions with fewer than four letters (such as "in"). The first and last words are always capitalized, regardless of what they are.
- A label should not be concluded with a semicolon.
- Be consistent within your application.

In general, the packages are self-contained. There are only two exceptions: You can reuse resources of other packages (see "Localization" on page 14) and you can reuse chooser definitions of other packages with the `apt:extendsChooser` attribute (see "Basic Data Elements: property, value, constraints" on page 39). Use this kind of references sparsely and deliberately. References to other packages introduce a dependency to that package and you cannot guarantee that every installation of the Desktop Manager contains the packages that you depend on.

As the Desktop Manager potentially scans every installed package for resource recovery, every resource key should be unique - not only inside your package, but also compared to the other packages. Either use the category hierarchy to prefix the local resource key or create your own hierarchical prefix, for example, by using a structure akin to the Java package structure or by using your product name.

The online help should be designed akin to the HTML files already provided in the standard packages. Include the following lines to allow for proper browser detection and CCS definitions:

```
<script type="text/javascript" src="/com_sun_web_ui/js/browserVersion.js">
</script>
<script type="text/javascript" src="/com_sun_web_ui/js/stylesheet.js">
/com_sun_web_ui/js/stylesheet.js"></script>
```

Use the styles "help-header-1", "help-header-2" and "help-header-3" for title layout.

◆ ◆ ◆ CHAPTER 5

Configuration Concepts

The presentation of the different trees in this document differs from what is covered in the administration guide. The administration guide does not mention the configuration tree because knowledge of the two different configuration and configuration profile trees is not necessary to use the Desktop Manager.

Strata

From the client point of view, the applications get configuration data from three separate data sources or *strata*. These strata are the default stratum, the user stratum and the profile stratum.

The user stratum and default stratum are the existing data sources that client applications currently deal with. The default stratum is deployed with the application and is mostly unchanged throughout its lifetime. It is stored locally alongside the application. The user stratum stores the changes made by a given user to the application settings. It is stored either locally or in a shared location.

The profile stratum is stored centrally in the configuration repository, which contains configuration settings that are managed by the Desktop Manager. These settings are associated on the server with elements such as organizations, roles, users, and hosts. They are accessed on behalf of a given user or host and are read-only for the user or host.

The Desktop Manager is able to read or write the configuration settings of the profile stratum only. The content of the default or user stratum cannot be accessed by the Desktop Manager. The client application configuration system is in charge of the retrieval and the combination of the values gathered from all strata. See "Merging" on page 31.

Trees

The Desktop Manager deals with four different hierarchical structures, also known as *trees*. To understand how the Desktop Manager works, it is important to distinguish these trees.

The first tree is the *organization tree*, the gray area inFigure 5–1, which represents the relationships between organizational units. The first level of the tree represents the organization itself. Subsequent levels can represent, for instance, departments and sub-departments. The last level can represent the members of these departments.

The second tree is the *domain tree*, which represents relationships between elements of the network such as domains or hosts. The first level of the tree represents the overall network. Subsequent levels can represent, for instance, the various subnets, and the last level the actual hosts in these subnets.

In the Desktop Manager, these two trees are currently obtained by interpreting the contents of an LDAP server, which is the typical repository for corporate structures. Each location within the tree in LDAP is called an *element*. Entries in a LDAP server are mapped to the elements recognized by the Desktop Manager, namely "Organization", "Role", "User", "Domain" and "Host".

The third tree is the *configuration tree*, which are represented by the blue areas in Figure 5–1. The configuration tree hierarchically groups configuration settings in the back end. At the highest level of the configuration tree are components. Components comprise configuration settings that configure one software component. All elements below a component are either nodes or properties. Nodes can contain nodes or properties. Properties contain configuration settings. Each configuration setting can be referred to by a path. For example, `org.openoffice.Office.Common/ExternalMailer/Program` is referring to the "Program" configuration setting, which is in the "External Mailer" node under the "Common" component.

Each element in the organization or domain tree can have its own configuration tree, resulting in two "trees of trees", one being an organization tree containing configuration trees and the other a domain tree containing configuration trees.

The fourth tree is the *configuration profiles tree*, which is represented by the yellow area of Figure 5–1. The configuration profiles tree is used to visually organize the configuration settings in order to conveniently browse and edit them. This is done by defining a hierarchy that is completely independent from the hierarchy in the configuration tree. The concrete values that are displayed in the configuration profile tree are obtained by referencing the location of the configuration settings in the configuration tree. See the arrows in Figure 5–1. This allows for the separation of the different design requirements for GUI and back-end data. For example, the position of a configuration setting changes more rapidly on the GUI than in the back end.

At the highest level of the configuration profile tree, there are applications with subsequent levels corresponding to various modules and sub-modules of that application, the last level being the actual configuration settings. A similar presentation can be seen in configuration systems dealing with many settings, such as the settings from StarOffice™ or Mozilla, where, for instance, the HomeUrl setting would be found under Mozilla/Navigator/HomeUrl in the Preferences dialog.

Note – The presentation of the different trees in this document differs from the presentation in the *Sun Desktop Manager 1.0 Administration Guide*. The administration guide does not mention the configuration tree because a knowledge of the two different configuration and configuration profile trees is not necessary to use the Desktop Manager.

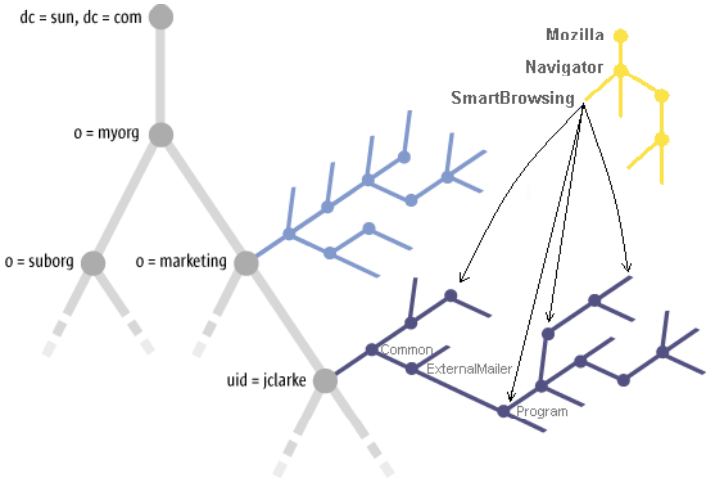


FIGURE 5–1 "Tree of trees"

Merging

The configuration settings that are finally used for a given element is determined by merging the configuration settings of that element and those of its parent elements on the client side. For instance, the settings for a user take into account the profiles assigned to that user and those assigned to the organizations that the user belongs to. The merging works by inheritance, that is, the user inherits the settings specified in the upper levels of the organization structure. This process is illustrated in Figure 5–2, which shows how the settings of the "marketing" organization are inherited by one of its members, user "jclarke". The configuration settings of user "jclarke" override some of the inherited settings.

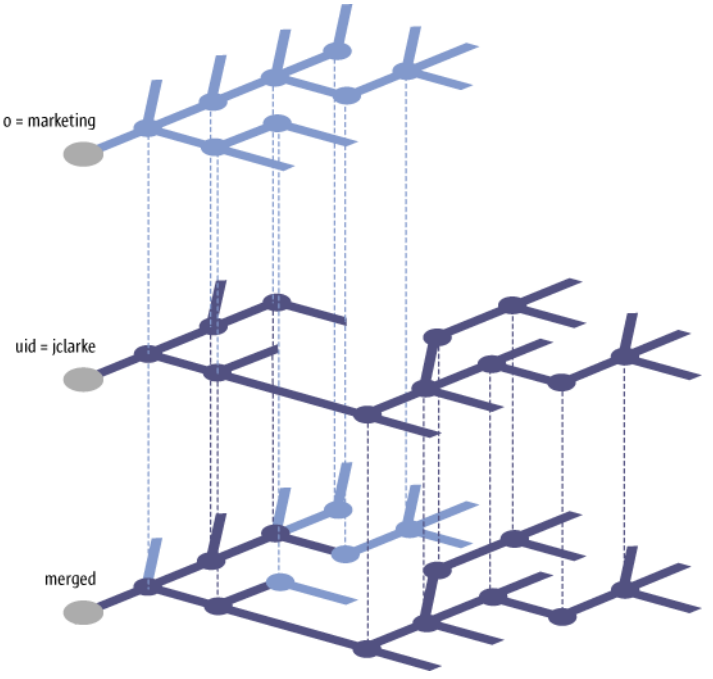


FIGURE 5–2 Merge

The three strata are merged to form the final set of configuration settings in the same way that the configuration settings are merged in the profile stratum. The user stratum takes precedence over the profile stratum, which in turn takes precedence over the default stratum. It is possible to mark configuration settings in the profile stratum so that configuration settings in the user stratum are no longer considered during the merge process, so that users not allowed to override settings made by an administrator with the Desktop Manager on their own client machines. This is called *protection*.

User-based and Host-based Configuration

The concepts for working with the organization tree and the domain tree are the same. The main difference between the two is that the organization tree consists of users and the domain tree consists of hosts. Having users and hosts in two separate trees enables the Desktop Manager to provide user-based and host-based configuration.

On the client side, the user-based configuration settings are fetched from the organization tree based on the user name. The host-based configuration settings are fetched from the domain tree based on the IP or the host name of the host the user is working on. The user settings are merged after the host settings, which means that the user settings take precedence over the host settings. For example, by offering these two types of configuration, roaming users can have one user-based configuration but nevertheless can make use of the optimal proxy configuration, depending on the host they are on.

◆ ◆ ◆ APPENDIX A

Configuration Path Mapping

One of the central design alignments of the Desktop Manager configuration repository is to be as flexible as possible to be able to act as repository for configuration data of as many other already existing configuration formats as possible.

The configuration format that is used in the repository partitions configuration data into *components*. Components are collections of configuration settings. The settings that belong to a component are typically used together and may be interrelated. Often they are associated with a particular client application, software module or application domain. Components are identified by their name and grouped into a hierarchy of packages by using structured names, for example `org.openoffice.Inet`

Each component has a hierarchical structure. This structure is composed of *nodes* and *properties*. A node is a structural element that serves as container for other nodes and properties. A property is a leaf element of the hierarchy. It contains one or more values. Nodes and properties are identified by their name, which must be unique within their parent node. That allows it to reference any node or property by its component and path, for example,
`org.openoffice.Inet/Settings/ooInetProxyType`

Other configuration systems have different configuration formats. The configuration data of other configuration systems is stored in the configuration registry using the APOC configuration format, but it has to be presented in the configuration format that the applications expect. A one-to-one mapping of the APOC format to the format expected by the applications is needed. The syntax mapping is silently done by the according APOC adapters. The only task remaining for template developers is to use the correct configuration path mappings when storing the configuration data in the configuration profile tree. These mappings are necessary to separate the `config` settings of the various client configuration systems in the central configuration repository. There are concrete mappings defined and considered by the adapters for the following configuration systems:

- StarOffice/OpenOffice Registry (OOR, used for StarOffice and OpenOffice.org)
- Gnome Configuration (GConf, used for Gnome applications)
- Java Preferences (used for Java programs)
- Mozilla Preferences (used for Mozilla)

StarOffice/OpenOffice Registry (OOR)

The OOR key naming scheme is the scheme used for the Desktop Manager configuration repository, therefore, no adaptation work is necessary for this configuration system.

Gnome Configuration (GConf)

To transform a GConf configuration element into an APOC component and path, the following mappings are performed:

- All GConf related components are prefixed with `org.gnome`.
- `/apps/<subdir>/...` is mapped to the component suffix `apps.<encoded subdir>`.
- `/desktop/<subdir>/...` is mapped to the component suffix `desktop.<encoded subdir>`.
- `/system/<subdir>/...` is mapped to the component suffix `system.<encoded subdir>`.
- `/extra/<subdir>/...` is mapped to the component suffix `extra.<encoded subdir>`.
- `/extra/<keyname>/...` is mapped to the component suffix `extra`.
- Keys not following the naming conventions are mapped to the component `suffix ooc.<encoded subdir>` if they feature a `subdir` and `ooc` otherwise.
- `/schemas/<keypath>` is mapped to the component part schemas, and then the above rules apply for the rest of the key.
- GConf key subdirs mapped to component parts are encoded to comply with component parts restrictions.

EXAMPLE A–1 Gnome Configuration

- `/apps/myapplication/sampleSub.Dir/sampleSetting` becomes `org.gnome.apps.myapplication/sampleSub.Dir/sampleSetting`,
- `/desktop/sampleDir/sampleSub.Dir/sampleSetting` becomes `org.gnome.desktop.sampleDir/sampleSub.Dir/sampleSetting`,
- `extra/sampleSetting` becomes `org.gnome.extra/sampleSetting`,
- `/sample.Dir/sampleSetting` becomes `org.gnome.ooc.sample.Dir/sampleSetting`,
- `/schemas/apps/gnome-setting/sampleSubDir/sampleSetting` becomes `org.gnome.schemas.apps.gnome-setting/sampleSubDir/sampleSetting`.

Java Preferences

To transform a Java Preferences node/key pair into an APOC component and path, the first three node path elements (or all node path elements if less than three are present) will be appended to `java.prefs` to form a component name and the remainder of the node path elements and the key will form the path. Only user preferences are considered.

EXAMPLE A–2 Java Preferences

- `node /com/sun/star/configuration, key someKey` becomes `java.prefs.com.sun.star/configuration/someKey`,
- `node /com/acme/widget, key someKey` becomes `java.prefs.com.acme.widget/someKey`,
- `node /sample.Dir, key someKey` becomes `java.prefs.sample.Dir/someKey`.

Mozilla Preferences

To transform a Mozilla configuration element into an APOC component and path, the first element (assuming the name contains more than one) is appended to `org.mozilla` to form a component name, and the rest of the name is used as a node path. Preferences with only one element are stored in `org.mozilla.ooc` under their respective names.

EXAMPLE A–3 Mozilla Preferences

- `mail.server.default.isSecure` becomes `org.mozilla.mail/server/default/isSecure`,
- `sampleSetting` becomes `org.mozilla.ooc/sampleSetting`.

◆ ◆ ◆ APPENDIX B

Element Dictionary

This appendix provides a reference description for every element and attribute possible in a template.

Header Elements: apt:template, resImport, helpImport

```
<!ELEMENT apt:template (resImport*, category)>
<!ATTLIST apt:template
    xmlns:apt CDATA #FIXED "http://www.sun.com/jds/apoc/2004/template"
    xmlns:oor CDATA #FIXED "http://openoffice.org/2001/registry"
    xmlns:xs  CDATA #FIXED "http://www.w3.org/2001/XMLSchema"
    xmlns:xsi CDATA #FIXED "http://www.w3.org/2001/XMLSchema-instance"
>

<!ELEMENT resImport EMPTY><!ATTLIST resImport
    apt:packagePath NMTOKEN #REQUIRED
>
```

The root element template has two sub-elements: `resImport` and `category`, which are described in "Structure Elements: category, page, section" on page 38.

The `resImport` element is used to import resource files. All resource keys of the imported resource bundle are announced to the template. You have to import the resource to use its resource keys in, for example, the `apt:label` attributes. See "Structure Elements: category, page, section" on page 38. The `apt:packagePath` attribute specifies the location of the resource file using a path. The delimiter is a dot ("."). The file postfix (`.properties`) must not be specified, as well as the ISO language code (ISO-639) and the ISO country code (ISO 3166). The root directory of the path is the `res` directory located below the `package` directory. See also "Localization" on page 14.

Structure Elements: category, page, section

```
<!ELEMENT category (category | page)>
<!ATTLIST category
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:inlineHelp NMTOKEN #IMPLIED
>
<!ELEMENT page ((section | set)+, xmlHandler*)><!ATTLIST page
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:inlineHelp NMTOKEN #IMPLIED
    apt:onlineHelp CDATA #IMPLIED
>
<!ELEMENT section (property+)>
<!ATTLIST section
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
>
```

The `category` element is used to define the unique position of a page in the configuration profile tree. Its first attribute is the `apt:name` attribute. The `name` attribute is used to define a unique name for an element. It facilitates better orientation in large templates and referencing elements.

The second attribute of the `category` element is `apt:scope`. The `scope` attribute specifies to which tree the configuration setting can be applied. If the scope is "user", the configuration setting is applied to the organization tree only. If the scope is "host", the configuration setting is applied to the domain tree only. If the scope is "global", the configuration setting is applied to both trees. The default setting is "global". The elements inherit the scope from their parent elements, except if an element defines its own scope. If an element has a "user" scope and a configuration profile tree attached to a domain tree is displayed in the Content Area, that element is not displayed to the user. The same holds true if an element has "host" scope and a configuration profile tree attached to an organization tree is displayed.

The third attribute of the `category` element is `apt:label`. The `label` attribute specifies the name of the element displayed to the user and supports localization. The string specified by the `label` attribute is searched in the resource bundles first. If a key matching the string is found, its value is displayed on the GUI. If the string has no matching key in any of the resource bundles, the string itself is displayed on the GUI. If no `label` attribute is specified, the string specified by the `name` attribute is rendered in the GUI. If both attributes are not defined, no output is rendered.

The fourth attribute of the `category` element is `apt:inlineHelp`. The `inlineHelp` attribute specifies the help text displayed on the GUI. The help is displayed to the right of the category names in the "Comment" column. It supports localization in the same manner as the `label` attribute described in the preceding paragraph.

There is exactly one page element at the end of a category hierarchy. This element represents one option page. It contains four attributes recognized by the `category` element: `name`, `scope`, `label`, and `inlineHelp`. The value of the `inlineHelp` attribute is displayed below the page title. The value of the `label` attribute is displayed as the page title. The category and page names define the unique location and name of a page in the configuration profile tree.

The `apt:onlineHelp` attribute is used to make the HTML file containing the online help available to the Desktop Manager. The HTML page referred to by this element is displayed as context-sensitive help if the user clicks the Help link in the masthead of the Desktop Manager. The `apt:filePath` attribute specifies the location of the help file using a path. The delimiter is a forward slash ("/"). The file postfix (`.html`) must not be specified, as well as the ISO language code (ISO-639) and the ISO country code (ISO 3166). The root directory of the path is the `web` directory located below the `package` directory. See also "Localization" on page 14.

A page can contain an arbitrary number of sections or sets, followed by an optional list of xmlHandlers. Thus the `page` element contains the sub-elements `section`, `set` (see "Dynamic Data Elements: set" on page 43) and `xmlHandler` (see "Interaction Elements: xmlHandler, event, action, choose, command" on page 45).

The `section` element creates a visual group of all its property sub-elements in a table-like layout. It contains four attributes recognized from the `category` element: `name`, `scope`, and `label`. The value of the `label` attribute is displayed as the section title.

Basic Data Elements: property, value, constraints

```
<!ELEMENT property (constraints?, value*, visual)>
<!ATTLIST property
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:inlineHelp NMTOKEN #IMPLIED
    apt:dataPath CDATA #REQUIRED
    oor:type (xs:boolean | xs:short | xs:int | xs:long | xs:double |
              xs:string | xs:hexBinary |
              oor:any | oor:boolean-list | oor:short-list | oor:int-list |
              oor:long-list | oor:double-list | oor:string-list | oor:hexBinary-list)
              #IMPLIED
    apt:storeDefault (true | false) #IMPLIED
    apt:xmlHandler IDREF #IMPLIED
    apt:extendsProperty CDATA #IMPLIED
>
<!ELEMENT visual (checkBox | chooser)?>
<!ATTLIST visual
 apt:type (textField | password | textArea | radioButtons | comboBox | stringList |
           colorSelector | hidden) #IMPLIED
>
```

```
<!ELEMENT checkBox EMPTY>
<!ATTLIST checkBox
 apt:labelPost NMTOKEN #IMPLIED
>

<!ELEMENT chooser EMPTY>
<!ATTLIST chooser
 apt:labelPopup NMTOKEN #IMPLIED
 apt:listDataPath CDATA #IMPLIED
 apt:extendsChooser CDATA #IMPLIED
>
<!ELEMENT constraints (enumeration*, length?, minLength?, maxLength?, minInclusive?,
                       maxInclusive?, minExclusive?, maxExclusive?)>
<!ELEMENT enumeration EMPTY>
<!ATTLIST enumeration
    oor:value CDATA #REQUIRED
    apt:label NMTOKEN #IMPLIED
>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value
    xsi:nil (true | false) #IMPLIED
    oor:separator CDATA #IMPLIED
>
```

The `property` element provides the visualization of configuration settings by way of GUI elements, such as checkboxes, radio buttons, and edit fields. It contains four attributes recognized by the `category` element: `name`, `scope`, `label` and `inlineHelp`. The inline help is displayed below the input fields (or in non-editable appearance below the value string) in the "value" column. The value of the label attribute is displayed as the label of the GUI elements. The category, page, section and property names define the unique location and name of a page in the configuration profile tree.

The attribute `apt:dataPath` defines a path that points to the location in the data back end that stores the value of the property. The value of the `dataPath` attribute is an absolute component path, for example, `org.openoffice.Office.Common/ExternalMailer/Program`. See Appendix A. The `dataPath` attribute of a `property` element must point to a data back-end property. Pointing to a data back-end node yields a runtime error.

The `apt:type` is used to specify the type of the repository configuration data. The following types are defined:

xs:boolean	Boolean value (true/false)
xs:short	16 - bit integer number
xs:int	32 - bit integer number

xs:long	64 - bit integer number
xs:double	Floating point number (value range as for IEEE 64-bit double)
xs:string	Plain Text (Sequence of printable Unicode characters)
xs:hexBinary	Sequence of uninterpreted octets, hex encoded
oor:any	Encompasses all of the types mentioned previously
oor:*-list	List of any of the typed mentioned previously

These types resemble the types defined in the StarOffice/OpenOffice registry (OOR) format. Wherever possible, the templates use the syntax of the OOR format for the sake of reusability. For more information on types, use the OpenOffice.org Registry Format (OOR) documentation at `http://util.openoffice.org/common/configuration/oor-document-format.html`

The attribute `apt:storeDefault` instructs the Desktop Manager to store the default data in the data back end. The default data is defined by the `value` element (see below) and used to display the default to the user. If the user doesn't change the value or explicitly requests storing the default data by executing the "Apply Default" action in the Content Area, the default data is not stored in the repository. By setting the value of the `storeDefault` attribute to true, the default data is stored even if the user does not change the value or executes "Apply Default".

A property element has three sub-elements: `constraints`, `value` and `visual`.

The visual element defines the visual type of the property on the GUI. The following visual types are recognized: `checkBox`, `radioButtons`, `comboBox`, `stringList`, `textField`, password, `textArea`, `chooser`, `colorSelector` and `hidden`. Every GUI element has two appearances: editable and non-editable. The non-editable appearance is rendered if the administrator using the Desktop Manager has no write privileges for that property.

The `hidden` property does not render a visual GUI element, but passes the value associated with the property to the browser in a hidden field. This feature proves useful, for example, if one value entered on the front-end has to be saved at more than one location in the back end.

The visual type is defined by the `apt:type` attribute of the `visual` element. There are two exceptions: `checkBox` and `chooser`. These two GUI elements need additional information in order to be displayed correctly, so they have their own sub-elements containing this information.

The checkbox property displays strings before and after the checkbox. It is represented by the `checkBox` element. Two additional strings are needed for displaying the `checkBox` GUI element in the non-editable appearance (see previous table). As a result, a `checkBox` GUI element needs four strings, which are displayed as follows:

1. In front of the checkbox. This string is defined in the `label` attribute of the `property` element.
2. After the checkbox. This string is defined in the `apt:labelPost` attribute of the `checkBox` sub-element. If that attribute is not defined, "`.post`" is appended to the string defined in the `label` attribute. This string is searched as key in the resource files.

3. Instead of a checked checkbox, if the checkbox is rendered in the non-editable appearance. The string is defined in the `label` attribute of the first `enumeration` sub-element of the `constraints` element. If no constraints are given, the postfix ".checked" is appended to the string defined in the `label` attribute of the `property` element. This string is searched as key in the resource files.
4. Instead of an unchecked checkbox, if the checkbox is rendered in the non-editable appearance. The string is defined in the `label` attribute of the second `enumeration` sub-element of the `constraints` element. If no constraints are given, the postfix ".unchecked" is appended to the string defined in the `label` attribute of the `property` element. This string is searched as key in the resource files.

The chooser property enables a value from a list of entries to be set. It is represented by the `chooser` element. In contrast to a combobox, the list of entries is editable. This list is stored in the back end at the location specified by the `apt:dataPath` attribute of the `chooser` element.

When the Edit button is clicked, a pop-up window opens, which provides a GUI for editing the list. The title of the content of the pop-up window is defined by the `apt:labelPopup` attribute of the `chooser` element. You can specify default values for the list by using the `enumeration` sub-element of the `constraints` element (see the paragraph dealing with the constraints below).

The `apt:extendsChooser` property of the `chooser` element is used to refer to another `chooser` element. This eases the reuse of previously defined `chooser` elements. All elements and attributes defined in that referred chooser are interpreted as if they were defined in the referring chooser. Sub-elements and attributes that are defined in the referring chooser overwrite elements and attributes of the chooser referred to. A property path is used to locate the referred chooser. A property path is a concatenation of the `apt:name` values of every element on the path from the root category to a property, delimited by forwards slashes ("/"). For example: `/StarOffice/Internet/Proxy/Settings/MyChooser`.

If the visual type is not specified, the GUI element used is derived from the `type` attribute. If the type is `xs:boolean`, a checkbox is used. If the type is a list type (e.g., `oor:short-list`), `xs:hexBinary` or `oor:any`, a text area is rendered. For all other types, an edit field is used. If neither the visual type nor the data type are specified, an edit field is rendered and the back-end data type is assumed to be `xs:string`.

The `constraints` element provides restrictions for the input fields. For example, if you want to allow a user to save only integer values between 1 and 5, providing an `oor:type` attribute with the value "xs:int" is not sufficient. You can specify the required restriction by specifying a minInclusive constraint of 1 and a maxInclusive constraint of 5.

There is a second application for the enumeration constraint if used with a `checkbox` property. The first `enumeration` `constraints` sub-element defines the value stored in the back end if the checkbox is checked, the second `enumeration` `constraints` sub-element defined the value stored in the back end if the checkbox is unchecked. If no constraint is given, the default values stored are true and false. The default strings displayed on the GUI in the non-editable appearance are "Enabled" and "Disabled".

The enumeration constraint has the same semantics for `radiobutton` and `combobox` properties as for `checkbox` properties, except that it is mandatory. The content of these elements is completely up to

the developer. The name displayed on the GUI is defined by the `label` attribute of the `constraint` element. It is possible to omit the `label` attribute in the enumeration constraint. In that case the additional resources are specified by postfixes appended to the string defined in the `label` attribute of the `property` element.

For example, assume a drop-down box, whose property label is "securityList" and whose enumeration constraints contain the values "1", "2" and "3" but the enumeration constraint labels are not defined. The strings displayed to the user are determined by searching the resources keys "securityList.1", "securityList.2" and "securityList.3" respectively.

The list entries of chooser properties are not localized. As a consequence the `apt:label` attribute of the enumeration constraint has no effect on these properties.

For a complete discussion of the other constraints, see the Property Constraints section of the OpenOffice.org Registry Format (OOR) documentation.

The `value` element contains the default value for the property. This default value may be the same value contained in the default stratum, or it may be introduced at this point. It defines the value that is displayed by the Desktop Manager if no data can be found in the data back end.

The definition of the `value` element is similar to the definitions given in the OOR format. The value element has no sub-elements, but rather three attributes: `nil`, `separator` and `lang`. If the `xsi:nil` attribute is set to true, the value of the property is defined as "has no value". The `oor:separator` attribute is used to specify the string that is used as separator for list tokens, if the value element contains list type values.

Tip – The list entries of `stringList` properties are stored as value of type `oor:string-list`. The default separator is a single space character.

For the `apt:xmlHandler` attribute, refer to "Interaction Elements: xmlHandler, event, action, choose, command" on page 45.

Dynamic Data Elements: set

```
<!ELEMENT set (page)>
<!ATTLIST set
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:labelPopup NMTOKEN #IMPLIED
    apt:dataPath CDATA #REQUIRED
    apt:elementNamePath CDATA #IMPLIED
>
```

Up to now, all elements handled static back-end content. The `set` element is used to handle dynamic content. It is, like the `section` element, a sub-element of a page and displays sets of properties in a table.

It contains three attributes recognized by the `category` element: `name`, `scope` and `label`. The value of the `label` attribute is displayed as the table title.

The `apt:dataPath` attribute defines a path that points to the data back-end node that contains the set of elements. The value of the `dataPath` attribute is an absolute path in the form of `org.openoffice.Office.Commands/Execute/Disabled` (see AppendixAppendix A) . The `dataPath` attribute of a `set` element must point to a back-end node. Pointing to a back-end property yields an error.

The `dataPath` attributes of descendants of the set must contain a dynamic part. This dynamic part specifies the location of the back-end nodes that are set members. The name of any back-end set member must be different to allow access to that member. To achieve this, variables are used.

Variables are prefixed with a dollar symbol: *$variable_name*. Valid variable names are `queriedId` and `silentId`. If the `$queriedId` variable is specified, the Desktop Manager displays an additional edit field that queries the user for a unique id. If the `$silentId` variable is specified, no ID is queried form the user; the Desktop Manager generates the unique ID by itself.

For example: The `dataPath` attribute of a `property` element has the value `org.openoffice.Office.Commands/Execute/Disabled/$queriedId/Command`. If the user creates a new set element, the user is additionally asked for the name of the set member. The concrete question string displayed on the GUI is specified by the `apt:labelPopup` attribute. If this attribute is omitted, the prompt `Please enter a name for the new entry.` is displayed.

It is possible to specify a relative path as the value of the `dataPath` attribute of sets or properties to minimize the length of the path strings. An example of a relative path is `./$queriedId/Command`. The absolute path is constructed by ascending the template element tree and prefixing the relative path with the `dataPaths` of its ancestors, until an absolute path is constructed. Assuming the parent of a property is a set. This set specifies the `dataPath` value `org.openoffice.Office.Commands/Execute/Disabled`. The Desktop Manager combines this path with the relative path `./$queriedId/Command`, resulting in the absolute path `org.openoffice.Office.Commands/Execute/Disabled/$queriedId/Command`.

It is not necessary to specify a `dataPath` attribute for the set, if all descendants of the set that support the `dataPath` attribute (`set` and `property` elements), specify an absolute path as the value of their `dataPath` attribute.

Recursive set structures (sets of sets) can also be handled. This is accomplished by having a set element containing a `page` element, which contains a `set` element again. The `dataPath` attribute of the sub set can specify a path relative to the super set.

If a set member is a back-end property, the value of the `oor:name` attribute of the back-end property is displayed as the label in the GUI. The value of the back-end property is displayed as the value of the GUI element.

If a set member is a back-end node, the value of the `oor:name` attribute of the back-end node is displayed as the name of the link. You can use the `apt:elementNamePath` attribute to override this naming scheme. The `elementNamePath` specifies a path relative to the back-end node. The path must

point to a back-end property and its value is displayed as the name of the link. If the user clicks on such a link, the Content Area is refreshed, displaying the page specified by the page sub-ement of the set.

Interaction Elements: xmlHandler, event, action, choose, command

```
<!ELEMENT xmlHandler (event+, action+)>
<!ATTLIST xmlHandler apt:name ID #REQUIRED>

<!ELEMENT event EMPTY>
<!ATTLIST event apt:type (onChange) #IMPLIED>

<!ELEMENT action (choose|command)+>

<!ELEMENT choose (when+, otherwise?)>

<!ELEMENT when (command+)>
<!ATTLIST when apt:test CDATA #REQUIRED>

<!ELEMENT otherwise (command+)>

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

The `xmlHandler` element is used to execute JavaScript code on the client side. The code execution is triggered depending on changes in the environment. These changes are propagated by events. Events are thrown by properties if their state changes. The type of an event indicates the type of the change.

An XML handler is defined by specifying the `xmlHandler` sub-element of the `apt:template` element. It requires a name defined by the `apt:name` attribute. An XML handler is registered to listen for events thrown by a property by specifying the `apt:xmlHandler` attribute of the `property` element using the name of the XML handler as its value.

The `event` element is used to specify the events the `xmlHandler` listens to by using its `apt:type` attribute. For now, only the `onChange` element is defined. This event is thrown by a property if its value is changed by the user. The value of a property changes atthe very moment the user changes its input, for example, by typing a key in an edit field, by deselecting a checkbox, or by choosing an entry in a list box. Moving the focus to, or removing the focus from, a GUI element does not trigger this event.

If a handler has registered for an event at one or more properties, and that event occurs for one of these properties, the code defined in the `action` element is executed. The `action` element contains at least one `choose` or `command` element.

A `command` element specifies the instructions to be executed on the client side. It can (up to now) contain assignments only. There are no calculations allowed, either on the left-hand side or the right-hand side of the assignment. Assignments obey the scheme: `<variable>=<value>`.

Variables follow a dotted notation: <property>.<qualifier>. The <property> must be the name of a property. The <qualifier> can have two values: “value” and “enabled”. The value qualifier denotes the value of that property. The value of a variable can be read and set to any value, as long that value is compatible with the type specified for the property. The "enabled" qualifier contains "true" if the property is enabled (can receive the focus) and "false" otherwise. It can be read and set to "true" or "false". Example: **propname.enabled=false**.

The choose element is similar to the choose element defined in XSLT, and allows the conditional execution of commands. It must contain at least one when element and may contain one otherwise element at the end.

The when element has one or more command sub-elements and one apt:test attribute. The test attribute must specify an expression evaluating to a Boolean value.

An expression may consist of variables, numbers, strings, and the following tokens:

=	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
( )	parentheses
not()	Boolean NOT
and	Boolean AND
or	Boolean OR
true	Boolean positive
false	Boolean negative

Example: **(propname.enabled!=false) and not(propname.value='foo')**.

If the expression evaluates to "true", the commands of the when element are executed and the choose statement is performed if it evaluates to "false" the next when element is evaluated. If none of the when elements is true and there is an otherwise element specified, the otherwise element commands are executed.

◆ ◆ ◆ APPENDIX C

The Template DTD

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

<!ELEMENT apt:template (resImport*, category)>
<!ATTLIST apt:template
    xmlns:apt CDATA #FIXED "http://www.sun.com/jds/apoc/2004/template"
    xmlns:oor CDATA #FIXED "http://openoffice.org/2001/registry"
    xmlns:xs  CDATA #FIXED "http://www.w3.org/2001/XMLSchema"
    xmlns:xsi CDATA #FIXED "http://www.w3.org/2001/XMLSchema-instance"
>
<!ELEMENT resImport EMPTY>
<!ATTLIST resImport
    apt:packagePath NMTOKEN #REQUIRED
>

<!ELEMENT category (category | page)>
<!ATTLIST category
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:inlineHelp NMTOKEN #IMPLIED
>

<!ELEMENT page ((section | set)+, xmlHandler*)>
<!ATTLIST page
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:inlineHelp NMTOKEN #IMPLIED
    apt:onlineHelp CDATA #IMPLIED
>

<!ELEMENT section (property+)>
<!ATTLIST section
    apt:name ID #REQUIRED
```

```
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
>

<!ELEMENT set (page)>
<!ATTLIST set
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:labelPopup NMTOKEN #IMPLIED
    apt:dataPath CDATA #REQUIRED
    apt:elementNamePath CDATA #IMPLIED
>

<!ELEMENT property (constraints?, value*, visual)>
<!ATTLIST property
    apt:name ID #REQUIRED
    apt:scope (user | host | global) #IMPLIED
    apt:label NMTOKEN #IMPLIED
    apt:inlineHelp NMTOKEN #IMPLIED
    apt:dataPath CDATA #REQUIRED
    oor:type (xs:boolean | xs:short | xs:int | xs:long | xs:double |
              xs:string | xs:hexBinary | oor:any | oor:boolean-list |
              oor:short-list | oor:int-list | oor:long-list | oor:double-list |
              oor:string-list | oor:hexBinary-list) #IMPLIED
    apt:storeDefault (true | false) #IMPLIED
    apt:xmlHandler IDREF #IMPLIED
    apt:extendsProperty CDATA #IMPLIED
>

<!ELEMENT visual (checkBox | chooser)?>
<!ATTLIST visual
    apt:type (textField | password | textArea | radioButtons | comboBox |
              stringList | colorSelector | hidden) #IMPLIED
>

<!ELEMENT  checkBox EMPTY>
<!ATTLIST checkBox
    apt:labelPost NMTOKEN #IMPLIED
>

<!ELEMENT  chooser EMPTY>
<!ATTLIST chooser
    apt:labelPopup NMTOKEN #IMPLIED
    apt:listDataPath CDATA #IMPLIED
>

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

```
<!ATTLIST value
    xsi:nil (true | false) #IMPLIED
    oor:separator CDATA #IMPLIED
>

<!ELEMENT constraints (enumeration*, length?, minLength?, maxLength?, minInclusive?,
                       maxInclusive?, minExclusive?, maxExclusive?)>

<!ELEMENT enumeration EMPTY>
<!ATTLIST enumeration
    oor:value CDATA #REQUIRED
    apt:label NMTOKEN #IMPLIED
>

<!ELEMENT length EMPTY>
<!ATTLIST length oor:value CDATA #REQUIRED
>
<!ELEMENT minLength EMPTY>
<!ATTLIST minLength oor:value CDATA #REQUIRED
>
<!ELEMENT maxLength EMPTY>
<!ATTLIST maxLength oor:value CDATA #REQUIRED
>
<!ELEMENT minInclusive EMPTY>
<!ATTLIST minInclusive oor:value CDATA #REQUIRED
>
<!ELEMENT maxInclusive EMPTY>
<!ATTLIST maxInclusive oor:value CDATA #REQUIRED
>
<!ELEMENT minExclusive EMPTY>
<!ATTLIST minExclusive oor:value CDATA #REQUIRED
>
<!ELEMENT maxExclusive EMPTY>
<!ATTLIST maxExclusive oor:value CDATA #REQUIRED
>
<!ELEMENT xmlHandler (event+, action+)>
<!ATTLIST xmlHandler apt:name ID #REQUIRED>

<!ELEMENT event EMPTY>
<!ATTLIST event apt:type (onChange) #IMPLIED>

<!ELEMENT action (choose|command)+>

<!ELEMENT choose (when+, otherwise?)>

<!ELEMENT when (command+)>
<!ATTLIST when apt:test CDATA #REQUIRED>
```

```
<!ELEMENT otherwise (command+)>
<!ELEMENT command (#PCDATA)>
```