

# **Configuring the DirXML Manual Task Service Driver (formerly Workflow Request)**

Implementation Guide

Document revision 0.2

Copyright © 2002-2004 Novell, Inc.

Work of Novell, Inc. All Rights Reserved.

## **General Disclaimer**

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. Novell, Inc., makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to

Overview .....	4
Modes of Operation.....	4
How E-Mail Messages and Web Pages Are Created by the Manual Task Service Driver.....	5
Templates .....	6
Replacement Tokens .....	8
Replacement Data.....	8
Template Action Elements .....	8
Subscriber Channel E-mail.....	9
Publisher Channel Web Server.....	10
Configuration.....	11
Driver Parameters.....	11
Driver Settings.....	11
Subscriber Settings .....	13
Publisher Settings .....	13
Subscriber Channel Policies .....	14
Subscriber Channel E-Mail Templates .....	15
Publisher Channel Policies .....	16
Publisher Channel Web Page Templates.....	16
Publisher Channel XDS Templates .....	17
Trace Settings .....	18
Appendix A – Replacement Data .....	19
Data Security .....	19
XML Elements .....	20
Appendix B – Automatic Replacement Data Items.....	24
Subscriber Channel Automatic Replacement Data.....	24
Publisher Channel Automatic Replacement Data.....	24
Appendix C – Template Action Elements Reference .....	26
Appendix D - <mail> Element Reference .....	29
Appendix E – Data Flow .....	32
Subscriber Channel Configuration .....	32
Publisher Channel Configuration .....	32
Step-by-Step Examination.....	32

Appendix F – Custom Element Handlers for the Subscriber Channel ..... 43

- Constructing URLs for use with the Publisher Channel Web Server ..... 43
- Constructing Message Documents using Stylesheets and Template Documents ..... 43
- SampleCommandHandler.java ..... 44

Appendix G – Custom Servlets for the Publisher Channel..... 45

- Using the Publisher Channel ..... 45
- Authentication ..... 45
- SampleServlet.java ..... 45

# Overview

This section provides an overview of the DirXML<sup>®</sup> Manual Task Service Driver including information necessary to successfully configure the Manual Task Service Driver.

The Manual Task Service Driver is designed to notify one or more users that a data event has occurred and in some cases that action is required on the users' part. In an employee provisioning scenario, the data event might be the creation of a new User object and the user action might include assigning a room number by entering data into Novell<sup>®</sup> eDirectory<sup>™</sup> or by entering data in an application. Other scenarios include notifying an administrator that a new user object has been created, notifying an administrator that a user has changed data on an object, etc.

Configuring the Manual Task Service Driver usually consists of configuring two separate but related subsystems: the Subscriber channel policies and e-mail templates, and the Publisher channel Web server templates (and possibly policies).

In addition, driver parameters such as SMTP server name, Web server port number, etc., must be configured.

## Modes of Operation

Two primary modes of operation are supported:

- **Direct Request for Data:** An e-mail message is sent requesting that a user enter data into eDirectory (possibly for consumption by another application). The e-mail recipient responds to the message by clicking a URL in the message. The URL points to the Web server running in the Publisher channel of the Manual Task Service Driver. The user then interacts with dynamic Web pages generated by the Web server to authenticate to eDirectory and to enter the requested data.
- **Event Notification:** An e-mail message is sent to a user without involving the Publisher channel. The e-mail message might simply be notification that something occurred in eDirectory, or it might be a request for data through a method other than the Publisher channel's Web server, such as Novell iManager, another application, or a custom interface.

### Example: Subscriber Channel E-Mail, Publisher Channel Web Server Response

An employee provisioning example scenario in which a new employee's manager assigns the employee a room number is as follows:

1. A new User object is created in eDirectory (for example, by the DirXML driver for the company's HR system).
2. The Manual Task Service Driver Subscriber sends an SMTP message to the user's manager and to the manager's assistant. The SMTP message contains a URL that refers to the Publisher channel Web server. The URL also contains data items identifying the user and identifying those authorized to submit the requested data.

3. The manager or the manager's assistant clicks the URL in the e-mail message. An HTML form appears in a Web browser. The manager or assistant then does the following:
  - Selects the DN for his or her eDirectory User object as a means of identifying who is responding to the e-mail message.
  - Enters his or her eDirectory password.
  - Enters the room number for the new employee.
  - Clicks the Submit button.
4. The room number for the new employee is submitted to eDirectory via the Manual Task Service Driver Publisher channel.

### Example: Subscriber Channel E-Mail, No Publisher Channel Response

An example scenario in which a new employee's manager assigns the employee a computer in an asset management system:

1. A new User object is created in eDirectory (for example, by the DirXML driver for the company's HR system).
2. The Manual Task Service Driver Subscriber sends an SMTP message to the user's manager and to the manager's assistant. The SMTP message contains instructions for entering data into the asset management system.
3. The manager or assistant enters data into the asset management system.
4. (Optional) The computer identification data is brought into eDirectory via a DirXML driver for the asset management system.

## How E-Mail Messages and Web Pages Are Created by the Manual Task Service Driver

E-mail messages, HTML Web pages, and XDS documents can all be considered *documents*. The Manual Task Service Driver creates documents dynamically, based on information supplied to the driver.

*Templates* are XML documents that contain the *boilerplate* or fixed portions of a document together with *replacement tokens* that indicate where the dynamic, or replacement, portions of the final, constructed document appear.

Both the Subscriber channel and the Publisher channel of the Manual Task Service Driver use templates to create documents. The Subscriber channel creates e-mail messages and the Publisher channel creates Web pages and XDS documents.

The dynamic portion of a document is supplied via *replacement data*. Replacement data on the Subscriber channel is supplied by the Subscriber channel policies (such as the Command Transformation policy). Replacement data on the Publisher channel is supplied by HTTP data to the Web server (both URL data and HTTP POST data). The Manual Task Service Driver may automatically supply certain data known to the Manual Task Service Driver (such as the Web server address).

The templates are processed by XSLT style sheets. These template-processing style sheets are separate from style sheets used as DirXML policies in the Subscriber or Publisher channels.

The replacement data is supplied as a parameter to the XSLT style sheet. The output of the style sheet processing is an XML, HTML, or text document that is used as the body of an e-mail message, as a Web page, or as a submission to DirXML on the Publisher channel.

Replacement data is passed from the Subscriber channel to the Publisher channel via a URL in the e-mail message. The URL contains a query portion that contains the replacement data items.

The Manual Task Service Driver ships with predefined style sheets sufficient to process templates in order to create e-mail documents, HTML documents, and XDS documents. Other custom style sheets can be written to provide additional processing options if desired.

An advanced method of creating documents is also available. This method uses only an XSLT style sheet and replacement data. No template is involved. However, this document assumes the template method is used because the template method is easier to configure and maintain without XSLT programming knowledge.

## Templates

This section describes document creation templates as used in the Manual Task Service Driver.

Templates are XML documents that are processed by a style sheet in order to generate an output document. The output document may be XML, HTML, or plain text (or anything else that can be generated using XSLT).

Templates are used to generate e-mail message text on the Subscriber channel, and to generate dynamic Web pages and XDS documents on the Publisher channel.

Templates contain text, elements, and replacement tokens. Replacement tokens are replaced in the output document by data supplied to the style sheet processing the template.

Several examples of templates for various purposes follow. In the examples, the replacement tokens are the character strings that are between two \$ characters and appear in **bold**.

Templates may also contain *action elements*. Action elements are control elements interpreted by the template-processing style sheet. Action elements are described in *Appendix C*. In the following examples action elements also appear in **bold**.

The following example template is used to generate an HTML e-mail message body:

```
<html xmlns:form="http://www.novell.com/dirxml/workflow/form">
<head></head>
<body>
Dear $manager$, <p/>
<p>
This message is to inform you that your new employee <b>$given-name$ $surname$</b> has
been hired.
<p>
You need to assign a room number for this individual. Click <a href="$url$">Here</a> to
do this.
</p>
<p>
Thank you, <br/>
HR Department
</p>
</body>
</html>
```

The following example template is used to generate a plain text e-mail message body:

```
<form:text xmlns:form="http://www.novell.com/dirxml/workflow/form">

Dear $manager$,

This message is to inform you that your new employee $given-name$ $surname$ has been
hired.

You need to assign a room number for this individual. Use the following link to do this:

$url$

Thank you,

The HR Department

</form:text>
```

The <form:text> element is required because templates must be XML documents. The <form:text> element is stripped as part of the template processing.

The following template is used to generate an HTML form used as a Web page for entering data:

```
<html xmlns:form="http://www.novell.com/dirxml/workflow/form">
  <head>
    <title>Enter room number for $subject-name$</title>
  </head>
  <body>
    <link href="novdocmain.css" rel="style sheet" type="text/css"/>
    <br/><br/><br/><br/>
    <form class="myform" METHOD="POST" ACTION="$url-base$/process_template.xml">
      <table cellpadding="5" cellspacing="10" border="1" align="center">
        <tr><td>
          <tr><td>
            <input TYPE="hidden" name="template" value="post_form.xml"/>
            <input TYPE="hidden" name="subject-name" value="$subject-name$"/>
            <input TYPE="hidden" name="association" value="$association$"/>
            <input TYPE="hidden" name="response-style sheet" value="process_template.xml"/>
            <input TYPE="hidden" name="response-template" value="post_response.xml"/>
            <input TYPE="hidden" name="auth-style sheet" value="process_template.xml"/>
            <input TYPE="hidden" name="auth-template" value="auth_response.xml"/>
            <input TYPE="hidden" name="protected-data" value="$protected-data$"/>
            You are:<br/>
            <form:if-single-item name="responder-dn">
              <input TYPE="hidden" name="responder-dn" value="$responder-dn$"/>
              $responder-dn$
            </form:if-single-item>
            <form:if-multiple-items name="responder-dn">
              <form:menu name="responder-dn"/>
            </form:if-multiple-items>
          </td></tr>
          <tr><td>
            Enter your password: <br/>
            <input name="password" TYPE="password" SIZE="20" MAXLENGTH="40"/>
          </td></tr>
          <tr><td>
            Enter room number for $subject-name$:<br/>
            <input TYPE="text" NAME="room-number" SIZE="20" MAXLENGTH="20"
            value="$query:roomNumber$"/>
          </td></tr>
          <tr><td>
            <input TYPE="submit" value="Submit"/> <input TYPE="reset" value="Clear"/>
          </td></tr>
        </table>
      </form>
    </body>
  </html>
```

The following template is used to generate an XDS document:

```
<nds>
  <input>
    <modify class-name="User" src-dn="not-applicable">
      <association>$association$</association>
      <modify-attr attr-name="roomNumber">
        <remove-all-values/>
        <add-value>
          <value>$room-number$</value>
        </add-value>
      </modify-attr>
    </modify>
  </input>
</nds>
```

## Replacement Tokens

The items delimited by \$ in the above example templates are replacement tokens. For example, \$manager\$ will be replaced by the replacement data named manager.

Replacement tokens can appear either in text or in XML attribute values (note the href value on the <a> element in the first example above).

## Replacement Data

Replacement data consists of strings that take the place of replacement tokens in the output document generated from a template. Replacement data is either supplied by Subscriber channel data, Publisher channel HTTP data, or it is supplied automatically by the driver. An additional type of replacement data is data retrieved from eDirectory via DirXML (query data).

Replacement data is more fully described in *Appendix A*.

**Subscriber channel data:** Subscriber channel replacement data is of two types. The first type is used as replacement values for replacement tokens in templates for creating e-mail messages. The second type is placed in the query portion of a URL so that the data is available for use on the Publisher channel when the URL is submitted to the Publisher's Web server.

**HTTP data:** Replacement data is supplied to the Publisher channel Web server and is supplied either as URL query string data, HTTP POST data, or both.

**Automatic data:** The Manual Task Service Driver supplies automatic data. Automatic data items are described in *Appendix B*.

**Query data:** Replacement tokens that start with `query:` are considered requests to obtain current data from eDirectory. The portion of the token that follows `query:` is the name of an eDirectory object attribute. The object to query is specified by one of the replacement data items `association`, `src-dn`, or `src-entry-id`. The items are considered in the order presented in the preceding sentence.

## Template Action Elements

Action elements are namespace-qualified elements in the template that are used for simple logic control or that are used to create HTML elements for HTML forms. The namespace used to qualify the elements is `http://www.novell.com/dirxml/workflow/form`. In this document and in the sample templates supplied with the Manual Task Service driver, the prefix used is `form`.

The elements appearing in **bold** in the above examples are action elements.

Action elements are described in detail in *Appendix C*.

## Subscriber Channel E-mail

The Subscriber channel of the Manual Task Service Driver is designed to send e-mail messages. To accomplish this, the driver supports a custom XML element named <mail>. Policies on the Subscriber channel construct a <mail> element in response to some eDirectory event (such as the creation of a user). An example <mail> element appears below:

```
<mail src-dn="\PERIN-TAO\novell\Provo\Joe">
  <to>JStanley@novell.com</to>
  <cc>carol@novell.com</cc>
  <reply-to>HR@novell.com</reply-to>
  <subject>Room Assignment Needed for: Joe The Intern</subject>
  <message mime-type="text/html">
    <stylesheet>process_template.xsl</stylesheet>
    <template>html_msg_template.xml</template>
    <replacement-data>
      <item name="manager">JStanley</item>
      <item name="given-name">Joe</item>
      <item name="surname">The Intern</item>
    <url-data>
      <item name="file">process_template.xsl</item>
      <url-query>
        <item name="template">form_template.xml</item>
        <item name="responder-dn protect="yes">\PERIN-TAO\big-org\phb</item>
        <item name="responder-dn protect="yes">\PERIN-TAO\big-org\carol</item>
        <item name="subject-name">Joe The Intern</item>
      </url-query>
    </url-data>
  </replacement-data>
  <resource cid="css-1">novdocmain.css</resource>
</message>
<message mime-type="text/plain">
  <stylesheet>process_text_template.xsl</stylesheet>
  <template>txt_msg_template.xml</template>
  <replacement-data>
    <item name="manager">JStanley</item>
    <item name="given-name">Joe</item>
    <item name="surname">The Intern</item>
  <url-data>
    <item name="file">process_template.xsl</item>
    <url-query>
      <item name="template">form_template.xml</item>
      <item name="responder-dn protect="yes">\PERIN-TAO\big-org\phb</item>
      <item name="responder-dn protect="yes">\PERIN-TAO\big-org\carol</item>
      <item name="subject-name">Joe The Intern</item>
    </url-query>
  </url-data>
</replacement-data>
</message>
<attachment>HR.gif</attachment>
</mail>
```

The Subscriber of the Manual Task Service Driver uses the information contained in the <mail> element to construct an SMTP e-mail message. A URL may be constructed and inserted into the e-mail message through which the e-mail recipient can respond to the e-mail message. The URL may point to the Publisher channel Web server or it may point to some other Web server.

The <mail> element and its content are described in detail in *Appendix D*.

## Publisher Channel Web Server

The Publisher channel of the Manual Task Service Driver runs a Web server configured so that users can enter data into eDirectory through a Web browser. The Web server is designed to work in conjunction with e-mail messages sent from the Subscriber channel of the Manual Task Service Driver.

The Publisher channel Web server can serve static files and dynamic content. Examples of static files are .css style sheets, images, etc. Examples of dynamic content are Web pages that change based on the replacement data contained in the URL or HTTP POST data.

The Publisher channel Web server is normally configured to allow a user to enter data into eDirectory in response to an e-mail that was sent by the Subscriber channel. A typical user interaction with the Web server is as follows:

1. The user submits the URL from the e-mail message to the Web server using a Web browser. The URL specifies the style sheet, template, and replacement data used to create a dynamic Web page (typically containing an HTML form).
2. The Web server creates an HTML page by processing the template with the style sheet and replacement data. The HTML page is returned to the user's Web browser as the resource referred to by the URL.
3. The browser displays the HTML page and the user enters the requested information.
4. The browser sends an HTTP POST request containing the entered information as well as other information that originated from the e-mail URL. The DN of the user responding to the e-mail and the user's password must be in the POST data.
5. The Web server authenticates the user using the user's DN and password. If the authentication fails then a Web page containing a failure message is returned as the result of the POST request. The failure message may be constructed using a style sheet and template specified in the POST data. If authentication succeeds then processing continues.
6. The Web server constructs an XDS document using a style sheet and template specified in the POST data. The XDS document is submitted to DirXML on the Publisher channel.
7. The result of the XDS document submission, together with a style sheet and template specified in the POST data, is used to construct a Web page indicating to the user the result of the data submission. This result Web page is sent to the browser as the result of the POST request.

# Configuration

This section describes configuring Manual Task Service Driver parameters and templates.

## Driver Parameters

### Driver Settings

This section describes parameters that appear in the “Driver Settings” section in the driver object user interface.

Many of these parameters are actually for the Publisher channel Web server. They appear under the Driver Settings area because the Manual Task Service Driver Subscriber also needs access to them.

### DN of the Document Base

This parameter is an eDirectory DN of a container object. The Manual Task Service Driver can load XML documents (including XSLT style sheets) from eDirectory as well as from disk. If XML documents should be loaded from eDirectory, this parameter identifies the root container from which documents are loaded.

Documents loaded from eDirectory reside in the attribute value of an eDirectory object. If unspecified, the attribute is XmlData. The attribute may be specified by appending a ‘#’ character followed by the attribute name to the name of the object containing the document.

For example, suppose that the document base DN is specified to be “novell\Workflow Documents” and that there is a container under “Workflow Documents” named templates.

If a DirXML-Style sheet object named “e-mail \_template” resides under the “templates” directory, then the following resource identifiers can be used to refer to the XML document: “templates/e-mail \_template” or “templates/e-mail \_template#XmlData”.

The resource identifiers can be supplied as replacement data, URL data, or HTTP POST data. For example, the following element might appear under a <message> element on the Subscriber channel:

```
<template>templates/e-mail _template#XmlData</template>
```

### Document Directory

This parameter identifies a file system directory that is used as the base directory for locating resources such as templates, XSLT style sheets, and other file resources served by the Publisher channel Web server. Example values are:

```
c:\Novell\Nds\workflow_files    Windows
SYS:\SYSTEM\workflow_files     NetWare
/usr/lib/dirxml/rules/workflow/workflow_files    Unix
```

## Use HTTP Server (true|false)

This parameter indicates whether the Publisher channel should run a Web server or not. This parameter should be set to true if the Web server should be run or false if the Web server should not be run.

If the Manual Task Service Driver is only to be used for sending e-mail with no response URL, or with a URL that points to another application, then the HTTP server should not be run to save system resources.

## HTTP IP Address or Host Name

This parameter allows you to specify on which of multiple, local IP addresses the Publisher channel Web server will listen for HTTP requests.

Leaving the HTTP IP address or host name parameter value blank will cause the Publisher channel Web server to listen on the default IP address. For servers with a single IP address this is sufficient. Placing a dot-notation IP address as the parameter value will cause the Publisher channel Web server to listen for HTTP requests on the address specified.

Note that the value specified for HTTP IP address or host name is used by the Subscriber channel mail handler to construct URLs if the host name or address is not specified in the mail command element. If the parameter Use HTTP server (true|false) is set to false then HTTP IP address or host name can be used to specify the address or name of a Web server to use in constructing URLs for mail messages.

## HTTP Port

This parameter is an integer value indicating which TCP port the Publisher channel Web server should listen on for incoming requests. If this value is not specified, the port number defaults to 80 or 443 depending on whether or not SSL is being used for the Web server connections.

If the Manual Task Service Driver is running on the DirXML server (that is, it is not being run under the DirXML Remote Loader on a remote machine) then the HTTP port should be set to something other than 80 or 443. This is because iMonitor or another process will typically be using ports 80 and 443.

## Name of KMO

If it is not blank, this parameter is the name of an eDirectory Key Material Object that contains the server certificate and key used for SSL by the Publisher channel Web server.

Setting this parameter causes the Publisher channel Web server to use SSL for servicing HTTP requests.

This parameter takes precedence over any Java keystore parameters (see below).

Using SSL is recommended for security reasons because eDirectory passwords are passed in HTTP POST data when using the Publisher channel Web server.

## Name of Keystore File

This parameter, together with `Keystore password`, `Name of certificate(key alias)`, and `Certificate password (key password)` are used to specify a Java keystore file that contains a certificate and key used for SSL by the Publisher channel Web server.

Setting this parameter causes the Publisher channel Web server to use SSL for servicing HTTP requests.

If `Name of KMO` is set, then this parameter and its associated parameters are ignored.

Using SSL is recommended for security reasons because eDirectory passwords are passed in HTTP POST data when using the Publisher channel Web server.

## Keystore Password

This parameter specifies the password for the Java keystore file specified with `Name of keystore file`.

## Name of Certificate (key alias)

This parameter specifies the name of the certificate to use in the Java keystore file specified with `Name of keystore file`.

## Certificate Password (key password)

This parameter specifies the password for the certificate specified using `Name of certificate (key alias)`.

## Subscriber Settings

### SMTP Server

This parameter specifies the name of the SMTP server that the Subscriber channel will use to send e-mail messages.

### SMTP Account Name

If the SMTP server specified using “SMTP server” requires authentication then this parameter specifies the account name to use for authentication. The password used is the “Application password” associated with the driver Authentication parameters.

### Default “From” Address

If specified, this is an e-mail address used in the SMTP from field for e-mail messages sent by the Subscriber channel. If this is not specified, then the `<mail>` elements sent to the Subscriber must contain a `<from>` element.

A `<from>` element under `<mail>` elements sent to the Subscriber overrides this parameter.

### Additional Handlers

If specified, then this is a whitespace-separated list of Java class names. Each class name is a custom class that implements the interface `com.novell.nds.dirxml.driver.workflow.CommandHandler` and handles a custom XDS element. (The handler for `<mail>` is a built-in handler).

Additional information about custom handlers is available in Appendix F.

## Publisher Settings

### Additional Servlets

If non-blank, this is a whitespace-separated list of Java class names. Each class name is a custom class that extends `javax.servlet.http.HttpServlet`. Custom servlets may be used to extend the functionality of the Publisher channel Web server.

Additional information about custom servlets is available in Appendix G.

## Subscriber Channel Policies

The configuration of the Subscriber channel policies depends on what a particular installation wants to accomplish with the Manual Task Service Driver. However, there are certain guidelines that may be helpful.

In general, the best place to construct a <mail> element to send to the Subscriber is in the Command Transformation policy. The reason for this is that most DirXML engine processing has been completed by the time commands reach the Command Transformation policy. This means that Create Policies have been processed for add events (allowing vetoing of add events for objects that don't have all the attributes necessary for constructing the e-mail, for example). This also means that modify events for objects without associations have already been converted to add events.

The XSLT style sheet that constructs the e-mail message may or may not need to query eDirectory for additional information.

For example, if the e-mail message is simply a welcome message to a new employee, then the add command can contain all the information necessary: Given Name, Surname, and Internet E-mail Address. This is accomplished by specifying in the Create policy that Given Name, Surname, and Internet E-mail Address are required attributes. This ensures that only add commands that contain the necessary information will reach the Command Transformation.

If, however, the e-mail message is a message to the manager of an employee, then the style sheet will need to query eDirectory. The manager DN can be obtained from the add event for the employee's User object, but a query must be made to obtain the manager's e-mail address because that information will be an attribute of the manager's User object.

In addition, if e-mail notifications are being generated as the result of modify commands for objects that are associated with the driver, then queries must be made to obtain information not contained in the modify command.

### Blocking Commands from Reaching the Subscriber

If e-mail messages are to be generated from events other than add events, then add events must be allowed to reach the Subscriber for those objects that are to be monitored. Allowing add events to reach the Subscriber will result in a generated association value being returned to DirXML from the Subscriber.

It is important that eDirectory objects to be monitored by the Manual Task Service Driver policies have an association for the Manual Task Service Driver. Only objects that have an association will have delete, rename, and move events reported to the driver. In addition, modify events on objects that do not have an association are converted to add events after the Subscriber channel event transformation.

All other commands (modify, move, rename, and delete) should be blocked by the Command Transformation policy and prevented from reaching the Subscriber. The Subscriber only handles <add> commands and <mail> commands. Other commands will result in the Subscriber returning an error.

## Generating E-Mail Messages

E-mail messages are sent by the Subscriber in response to receiving a <mail> element that describes the e-mail message to be sent. See *Appendix D* for a description of the <mail> element and its content.

E-mail messages can be generated in response to any DirXML event (add, modify, rename, move, delete).

The replacement data that is supplied with the <message> element children of a <mail> element will depend on two primary factors:

- The template used to generate the message body. Replacement items to be used by the e-mail template appear as children of the <replacement-data> element.
- The information needed by the Web page templates on the Publisher channel if the e-mail is to result in a response on the Publisher channel. Replacement items to be used by the Web page templates appear as children of the <url-query> element, which is a child of <url-data>, which in turn is a child of <replacement-data>.

If the e-mail message should contain a URL that points to the Publisher channel Web server and is used to solicit information from a user, then the replacement data must contain at least one `responder-dn` item. The values of the `responder-dn` items must be the DNs of the User objects of the users to which the message is being sent.

If a query replacement token (see *Replacement Data* under *Overview*) is used in the template, then the replacement data for the <message> element must contain an item named `src-dn`, `src-entry-id`, or `association` with the appropriate value. An association item can only be used if the eDirectory object to be queried already has an association for the Manual Task Service Driver. The association generated by the Subscriber for unassociated objects cannot be used because it will not yet have been written to the eDirectory object when the query takes place.

The <message> element may specify the MIME type of the message body. If the MIME type is specified but a style sheet is not specified (that is, there is no <stylesheet> element child of <message>) then one of two default style sheet names is used. If the MIME type is text/plain then the default style sheet name is `process_text_template.xml`. If the MIME type is anything other than text/plain then the default style sheet name is `process_template.xml`.

## Subscriber Channel E-Mail Templates

E-mail templates are XML documents containing boilerplate and replacement tokens. E-mail templates are used to generate e-mail message body text. See *Templates* under the *Overview* section for general information about templates.

The replacement tokens used in an e-mail template dictate the <item> elements that must be supplied as children of the <replacement-data> element that is constructed by the Subscriber channel policy that constructs the <mail> element. For example, if the e-mail template has the replacement token `$employee-name$` then there must be an <item name="employee-name"> element in the replacement data for the <message> element. If the employee name item is not present, then the resulting e-mail message body will have no text in the location occupied by the replacement token in the template.

E-mail templates can be used to generate message bodies that are plain text, HTML, or XML.

If an e-mail template generates a plain text message, then it must be processed by a style sheet that specifies plain text as its output type. If the style sheet does not specify plain text as its output type, then undesirable XML escaping will occur. The default Manual Task Service Driver style sheet named `process_text_template.xml` is normally used for processing templates that result in plain text.

## Publisher Channel Policies

In most implementations of the Manual Task Service Driver, no Publisher channel policies are needed. This is because it is possible to construct the Web page and XDS templates so that they result in exactly the XDS required and the XDS need not be further processed by policies.

If policies are required they will be very specific to an installation.

## Publisher Channel Web Page Templates

Web page templates are XML documents containing boilerplate and replacement tokens. Web page templates are used to generate Web page documents (typically HTML documents). See *Templates* under the *Overview* section for general information about templates.

Replacement tokens in Web page templates dictate what replacement data is supplied to as URL query data on the Subscriber channel. Replacement data on the Publisher channel is obtained from the URL query string for HTTP GET requests and from the URL query string and the POST data for HTTP POST requests.

As an example of the flow of replacement data from the Subscriber channel, to the e-mail message, and then to the Publisher channel Web server, consider the following:

The Manual Task Service Driver is configured so that a new employee's manager is asked to assign a room number to the new employee. The trigger for the e-mail to the manager is the <add> command for a new User object that is processed by the Subscriber channel Command Transformation policy.

When the manager clicks a URL in the e-mail message, a Web page is displayed in the manager's Web browser. The Web page must indicate for whom the manager is entering a room number.

To accomplish this, the <url-query> element on the Subscriber channel contains a replacement data item that identifies the new user by name:

```
<item name="subject-name">Joe the Intern</item>
```

This causes the URL query string to contain (among other things) "subject-name=Joe%20the%20Intern". (The "%20" is a URL-encoded space).

The manager's Web browser submits the URL to the Publisher channel Web server when the manager clicks the URL in the e-mail message. The Web server constructs a replacement data item named subject-name with the value Joe the Intern.

The Web page template also specified by the URL contains a replacement token \$subject-name\$. When the Web page template is processed by the style sheet to construct the Web page, the replacement token is replaced by Joe the Intern, which customizes the Web page for the employee whose User object creation caused the e-mail to be sent.

For additional information on a complete Subscriber channel to Publisher channel transaction, see *Appendix E*.

## Publisher Channel XDS Templates

XDS templates are XML documents containing boilerplate and replacement tokens. XDS templates are used to generate XDS documents that are submitted to DirXML on the Manual Task Service Driver's Publisher channel. See *Templates* under the *Overview* section for general information about templates.

Replacement tokens in XDS templates dictate some of the replacement data that is supplied to the Web server as data in an HTTP POST request.

For example, consider the following XDS template:

```
<nds>
  <input>
    <modify class-name="User" src-dn="not-applicable">
      <association>$association$</association>
      <modify-attr attr-name="roomNumber">
        <remove-all-values/>
        <add-value>
          <value>$room-number$</value>
        </add-value>
      </modify-attr>
    </modify>
  </input>
</nds>
```

The replacement tokens in the template dictate that the HTTP POST data must supply an association value and a room-number value.

Normally the association value would originate in the Subscriber channel. The Subscriber channel e-mail would place `association=some value` in the query string of the URL that is placed in the e-mail message. The Web page template used to generate the Web page when the URL is submitted to the Web server would typically place the association value in a hidden INPUT element:

```
<INPUT TYPE="hidden" NAME="association" VALUE="$association$"/>
```

Placing the association value as a hidden INPUT element will cause the “`association=some value`” pair to be submitted as part of the HTTP POST data.

The room-number value is entered in the Web page using an INPUT element similar to the following:

```
<input TYPE="text" NAME="room-number" SIZE="20" MAXLENGTH="20"/>
```

If the manager enters 1234 and clicks Submit, the Web browser sends “`room-number=1234`” as part of the HTTP POST data.

The Web server then generates an `<item name="association">` replacement data item and an `<item name="room-number">` replacement data item which are used when processing the XDS template.

The XDS document is generated by processing the XDS template with the style sheet specified in the POST data. Then the XDS document is submitted to DirXML on the Manual Task Service Driver's Publisher channel.

## Trace Settings

The Manual Task Service Driver outputs more verbose messages with various trace levels:

Level	Trace Message Description
0	No trace messages
1	Single line messages tracing basic operation
2	No additional messages (DirXML Engine traces XML documents at this level and above)
3	No additional messages
4	Messages relating to document construction from templates and style sheets
5	Replacement data documents traced

# Appendix A – Replacement Data

Replacement data is used with XML documents used as templates to construct e-mail messages, Web pages, and XDS documents. The actual replacement is accomplished by processing the template document with an XSLT style sheet that performs the replacement as part of constructing the output document.

Replacement data is supplied to the Manual Task Service Driver through different mechanisms on the Subscriber and Publisher channels.

## Subscriber Channel

- Replacement data is supplied as part of the <mail> element.
- Part of the supplied replacement data may be URL data. If URL data is supplied, it is processed and completed and replaced by automatic data items (see Appendix B).
- If the <mail> element specifies that an association value should be constructed (that is the <mail> element has a *src-dn* attribute) then an automatic data item named “association” is added to the replacement data.

## Publisher Channel

- Replacement data is supplied in the HTTP URL data and HTTP POST data.
- Automatic URL replacement data items are added to the replacement data before it is used in template processing.

Replacement data is presented during template processing as an XML document. The replacement data document is passed to the style sheet processing the template as a parameter named replacement-data. If no template is used, the XML document is processed directly by the style sheet.

# Data Security

Data items are passed from the Subscriber channel to the Publisher channel via a URL contained in the e-mail sent by the Subscriber channel. Changing certain data items in the URL represents a security threat. For example, if the responder-dn values in the URL supplied by the Subscriber channel in the URL are replaced by another user’s DN in the URL submitted to the Publisher channel Web server, it would allow an unauthorized user to change data in eDirectory.

To ensure that the data in the submitted URL is the same as the data originally supplied by the Subscriber channel, *protected data* is provided. Protected data is data that cannot be changed for security reasons. These will vary by configuration but will always include the responder-dn data items, and data items corresponding to any eDirectory objects whose values are to be changed.

Data items are protected by encrypting the original values and placing the encrypted values into a URL query string. When the Publisher Web server receives the encrypted values, the Publisher decrypts the values and uses them to compare the unencrypted data items that are supplied by an HTTP GET or POST request.

If an instance of a data item appears in the encrypted data, then an unencrypted data item value must match one of the encrypted data item values. If the unencrypted data item value does not match one of the encrypted data item values, then the HTTP request is rejected by the Publisher channel Web server.

In addition, any HTTP POST request that does not contain protected data is rejected.

## Example

In an HTTP POST request, the Publisher channel Web server uses the unencrypted POST data named responder-dn to check the password supplied by the POST data. This is done to authenticate the responding user against the user's eDirectory object.

Suppose the Subscriber channel <url-query> element content specifies two data items as follows:

```
<item name="responder-dn" protect="yes">\PERIN-TAO\novell\phb</item>
<item name="responder-dn" protect="yes">\PERIN-TAO\novell\carol</item>
```

The URL generated by the Subscriber channel will contain both responder-dn values in the protected data.

Suppose a malicious user obtains the URL that is generated and sent in an e-mail message. The malicious user uses the URL to obtain the HTML form that allows users to change data for an eDirectory object.

In the HTTP POST request that is submitted to the Web server, the malicious user uses his eDirectory DN (responder-dn=\PERIN-TAO\novell\wally) as the unencrypted responder-dn value. The malicious user also submits his own password in the POST data so that the authentication that the Web server performs will succeed.

However, when the Publisher channel Web server receives the HTTP POST data it will fail to find "\PERIN-TAO\novell\wally" in the encrypted protected data and will reject the POST request.

## XML Elements

The elements that make up a replacement data document are described below. If no XML attributes are described for an element, then none are allowed.

### <replacement-data>

The <replacement-data> element can appear in the following locations:

- 1) As a child of the <message> element under a Subscriber channel <mail> element. The Manual Task Service Driver processes the supplied <replacement-data> element into a standalone <replacement-data> element for use in template processing. The following processing occurs:
  - a) If an association value is created for the enclosing <mail> element an <item name="association"> element is added to the replacement data. The value of the created element is the association value that will be returned to DirXML.
  - b) If the <replacement-data> element has a <url-data> element child, then the <url-data> element is replaced by several <item> elements that contain constructed URL data. See <url-data> and <url-query>.
- 2) As the standalone, top-level element of a replacement data document used when constructing a document using a style sheet on either the Subscriber or the Publisher channels.

## <item>

The <item> element can be a child of the <replacement-data> element, the <url-data> element, or the <url-query> element. The content of the <item> element is the text used in the substitution of replacement tokens in templates. <item> elements are always named using the *name* attribute.

### <item> attributes

**name:** The value of the *name* attribute specifies the name by which this data item is referenced by replacement tokens. For example, if the value of the *name* attribute is *manager*, then the replacement token *\$manager\$* will be replaced by the value contained by <item name="manager"> element. The *name* attribute is required.

**protect:** For <item> elements that are children of <url-query> elements, the *protect* attribute specifies whether the item will be added to the protected data section of the URL query string (see <url-query>). If the *protect* attribute is present, it must have the value *yes*.

### Predefined <item> names

Certain <item> elements have predefined meanings to either the Subscriber channel, the Publisher channel, or both channels. These are described here:

**template:** The Publisher channel treats the value of the template item as the name of the template document to use in generating the response to an HTTP GET request.

When <item name="template"> appears as a child of the <url-query> element on the Subscriber channel, the value is placed into the URL query data to specify to the Publisher channel Web server the name of the template document to use when responding to the HTTP GET request.

**responder-dn:** The Publisher channel uses the value of the responder-dn item in HTTP POST data as the DN of the eDirectory object against which the password supplied in the HTTP POST data is validated.

The Web server will reject any HTTP POST request that does not contain a responder-dn value and a password value. In addition, if the HTTP POST data does not contain a protected-data item, then the request will be rejected.

The Subscriber channel supplies one or more <item name="responder-dn" protect="yes"> elements under the <url-query> element. Because the responder-dn items are used for user authentication, the items must be protected.

**password:** Supplied to the Publisher channel Web server via HTTP POST data. The item content is the password which is validated against the eDirectory object specified by the responder-dn item in the POST data. The password item is normally entered in the HTML form used to generate the HTTP POST request.

Example:

```
<INPUT TYPE="password" NAME="password" SIZE="20" MAXLENGTH="40"/>
```

**response-template:** Supplied to the Web server via HTTP POST data. Used to generate the Web page used as the response to the POST. The response-template item is normally specified using a hidden INPUT element in the HTML form used to generate the HTTP POST request.

Example:

```
<INPUT TYPE="hidden" NAME="response-template" VALUE="post_form.xml"/>
```

**response-stylesheet:** Supplied to the Web server via HTTP POST data. Used to generate the Web page used as the response to the POST. The response-stylesheet item is normally specified using a hidden INPUT element in the HTML form used to generate the HTTP POST request.

Example:

```
<INPUT TYPE="hidden" NAME="response-stylesheet" VALUE="process_template.xml"/>
```

**auth-template:** Supplied to the Web server via HTTP POST data. Used to generate the Web page that is used as the response to the POST if authentication of the user fails. The auth-template item is normally specified using a hidden INPUT element in the HTML form used to generate the HTTP POST request.

Example:

```
<INPUT TYPE="hidden" NAME="auth-template" VALUE="auth_response.xml"/>
```

**auth-stylesheet:** Supplied to the Web server via HTTP POST data. Used to generate the Web page that is used as the response to the POST if authentication of the user fails. The auth-template item is normally specified using a hidden INPUT element in the HTML form used to generate the HTTP POST request.

Example:

```
<INPUT TYPE="hidden" NAME="auth-stylesheet" VALUE="process_template.xml"/>
```

**protected-data:** The protected-data item contains the encrypted data constructed by the Subscriber channel. On the Subscriber channel, the protected data item is an automatically-supplied item.

On the Publisher channel, the protected-data item is obtained from the URL query string for an HTTP GET request and is obtained from the POST data for an HTTP POST request.

The protected data item is typically passed from the HTTP GET request into the Web page used to generate the HTTP POST via a replacement token in the template used to construct the response to the HTTP GET.

Example:

```
<INPUT TYPE="hidden" NAME="protected-data" VALUE="$protected-data$"/>
```

## <url-data>

The <url-data> element is a child of the <replacement-data> element found under the <message> element on the Subscriber channel. It contains <item> elements used to construct the URL and related data items that are supplied to the template used in constructing the e-mail message. It also contains the <url-query> element.

For the purposes of the Manual Task Service driver, URLs consist of five parts:

1. A scheme such as http, https, ftp, etc.
2. A host such as www.novell.com or 192.168.0.1.
3. A port number. This is a colon followed by a decimal integer. For example, :80 or :8180.
4. A file or resource specifier. This is typically a filename and may include path information. For example, stylesheets/process\_template.xml.
5. A query string. This is a collection of name-value pairs, separated by & characters. For example, template=form\_template.xml&protected-data=AabABJKEL=

### Predefined <item> Names Under <url-data>

<item> elements under the <url-data> element are ignored unless they are one of the following. All of them are optional.

**file:** Specifies the file portion of the URL. If used with the Publisher channel Web server, then the file item specifies the style sheet to use to construct the initial HTML page returned in response to

the URL. If used with a server other than the Publisher channel Web server, then the file item specifies the name of the resource that the URL will refer to.

If the file item does not appear, the URL file portion defaults to process\_template.xml.

**scheme:** Optional item found under the <url-data> element. If present, specifies the scheme portion of the URL (such as http or ftp). The scheme item would typically be used only if the URL were to point at a server other than the Publisher's Web server.

If the scheme item does not appear, the URL scheme defaults to either http or https, depending on the configuration of the Publisher channel Web server.

**host:** Optional item found under the <url-data> element. If present, specifies the host portion of the URL. The host item would typically be used only if the URL were to point at a server other than the Publisher's Web server.

If the host item does not appear, the URL host defaults to the IP address of the server on which the Manual Task Service Driver is running (that is the IP address of the Publisher channel Web server).

**port:** Optional item found under the <url-data> element. If present, specifies the port portion of the URL. The port item would typically be used only if the URL were to point at a server other than the Publisher's Web server.

If the port item does not appear, the URL port defaults to the port on which the Publisher channel Web server is running.

### <url-query>

The <url-query> element is a child of the <url-data> element. It contains <item> elements that are used to construct the query portion of the URL used in the e-mail message.

Each item that appears as a child of the <url-query> element is placed in the query string in the form name="value" where name is the value of the <item> element's *name* attribute and value is the string content of the <item> element.

Item elements that appear under <url-query> may have a *protect* attribute with the value "yes." If this is the case, then the item names and values will be encrypted and placed within a generated name-value pair in the URL query string. The name of the generated value is protected-data. The value is the Base64 encoded and encrypted name-value pair or pairs for multivalued attributes.

Protecting data ensures that the data cannot be changed when the URL is submitted to the Publisher channel Web server. For example, the responder-dn data items need to be protected to ensure that only those users authorized to respond to the e-mail message are able to change eDirectory data.

If the URL generated is to be used with the Publisher channel Web server, the <url-query> element must contain at least one <item name="responder-dn" protect="yes"> element or the Web server will reject the eventual HTTP POST request.

## Appendix B – Automatic Replacement Data Items

The Manual Task Service Driver automatically supplies certain replacement data item elements. This section describes those data items.

### Subscriber Channel Automatic Replacement Data

The following data items are added automatically to replacement-data documents during processing by the Subscriber channel:

**association:** An `<item name="association">` element is added to the replacement-data document if the `<mail>` element has an `<association>` element child, or if the Subscriber will return an `<add-association>` element. The content of the `<item>` element is the association value for the eDirectory object that is associated with the e-mail message being processed. The association value may not yet be written to the eDirectory object; therefore, the association value cannot be used in queries.

**url:** The content of the `<item>` element is the complete URL to be used in the e-mail message. On the Subscriber channel, the url item is created from the following items found under the `<url-data>` element: scheme, host, port, file, and the items underneath the `<url-query>` element. If scheme, host, or port are not found, then default values are used. The default values are determined from the configuration of the Publisher channel Web server.

**url-base:** The content of the `<item>` element is the portion of the generated URL not including the resource identifier (file) and not including the query string.

**url-query:** The content of the `<item>` element is a URL query string generated from `<item>` elements underneath the `<url-query>` element.

**url-file:** The content of the `<item>` element is the resource identifier for the URL.

**protected-data:** The content of the `<item>` element is an encrypted form of name-value pairs obtained from `<item>` elements under the `<url-query>` element. Only `<item>` elements whose *protect* attribute is set to “yes” are added to the protected data value. See *Data Security* in Appendix A for more information about protected-data.

### Publisher Channel Automatic Replacement Data

The following data items are added automatically to replacement-data documents during processing by the Publisher channel Web server:

**post-status:** An `<item name="post-status">` element is created and added to the replacement-data document by the Publisher channel Web server during the processing of an HTTP POST request. An HTTP POST request to the Web server is a request to submit an XDS document to DirXML. DirXML returns a status document as the result of the XDS submission. The content of the `<item name="post-status">` element is the value of the *level* attribute of the `<status>` element that is returned by DirXML as the result of the submission to DirXML.

The post-status item is typically used in the construction of the Web page that is returned as the result of the HTTP POST request.

**post-status-message** An `<item name="post-status-message">` element is created and added to the replacement-data document by the Publisher channel Web server during the processing of an HTTP POST request. An HTTP POST request to the Web server is a request to submit an XDS document to DirXML. DirXML returns a status document as the result of the XDS submission. The content of the `<item name="post-status-message">` element is the content of the `<status>` element that is returned by DirXML as

the result of the submission to DirXML. The post-status-message item will only be created if the <status> element returned by DirXML has content.

The post-status-message item is typically used in the construction of the Web page that is returned as the result of the HTTP POST request.

**url** An <item name="url"> element is created and added to the replacement-data document by the Publisher channel Web server during processing of HTTP GET and HTTP POST requests . The <item> element is added before using the replacement-data document to construct any documents. The URL scheme, host, and port are determined by the Web server configuration.

**url-base** An <item name="url-base"> is created and added to the replacement data document by the Publisher channel Web server during processing of HTTP GET and HTTP POST request. The <item> element is added before using the replacement-data document to construct any documents. The content of the url-base <item> element on the Publisher channel is the same as the url <item> element.

# Appendix C – Template Action Elements Reference

Action elements are namespace-qualified elements in a template document that are used for simple logic control or are used to create HTML elements for HTML forms. The namespace used to qualify the elements is `http://www.novell.com/dirxml/workflow/form`. In this document and in the sample templates supplied with the Manual Task Service driver the prefix used is `form`.

Any action element not specifically covered in this appendix is stripped from the output document by the template-processing style sheet (unless the style sheet is customized). This behavior allows, for example, the use of a `form:text` element to enclose the data for a plain text e-mail message, thereby making the template valid XML.

## <form:input>

The `form:input` element is used to generate one or more HTML INPUT elements based on the presence of one or more replacement data items. The number of INPUT elements created corresponds with the number of replacement data items with the name specified by the `form:input` element's "name" attribute.

### Attributes

**Name:** Specifies the name of the replacement data items that are used to create the INPUT elements. The attribute value is used as the value of the *name* attribute of the created INPUT elements.

**type** or **TYPE:** Specifies the value of the *type* attribute of the created INPUT elements.

**value:** If the *value* attribute's value is equal to "yes," then a *value* attribute is added to the created INPUT elements whose value is the string value of the replacement data item. If the *value* attribute's value is other than "yes," then the content of the created INPUT elements is set to the string value of the replacement data item.

### Example

```
<form:input name="responder-dn" TYPE="hidden" value="yes"/>
```

will create one or more INPUT elements similar to

```
<INPUT name="responder-dn" TYPE="hidden" value="\PERIN-TAO\novell\phb"/>
```

## <form:if-item-exists>

The form:if-item-exists element is used to conditionally insert data into the output document. The content of form:if-item-exists is processed only if the specified item appears in the replacement data.

### Attributes

**Name:** Specifies the name of the replacement data item. If one or more examples of the replacement data item exist, then the contents of the form:if-item-exists element are processed.

### Example

```
<form:if-item-exists name="post-status-message">
  <tr>
    <td>
      Status message was: $post-status-message$
    </td>
  </tr>
</form:if-item-exists>
```

The above example inserts a row into an HTML table only if there is a replacement data item named post-status-message.

## <form:if-multiple-items>

The form:if-multiple-items element is used to conditionally insert data into the output document. The content of form:if-multiple-items is processed only if the specified item appears more than once in the replacement data.

### Attributes

**name:** Specifies the name of the replacement data item. If more than one example of the replacement data item exists, then the content of the form:if-multiple-items is processed.

### Example

```
<form:if-multiple-items name="responder-dn">
  <form:menu name="responder-dn" />
</form:if-multiple-items>
```

The above example will build an HTML SELECT element (see <form:menu>) if there are more than one replacement data items with the name responder-dn.

## <form:if-single-item>

The form:if-single-item element is used to conditionally insert data into the output document. The content of form:if-single-item is processed only if the specified item appears exactly once in the replacement data.

### Attributes

**name:** Specifies the name of the replacement data item. If the named item appears exactly once in the replacement data, then the content of the form:if-single-item is processed.

### Example

```
<form:if-single-item name="responder-dn">
  <input TYPE="hidden" name="responder-dn" value="$responder-dn$" />
  $responder-dn$
</form:if-single-item>
```

The above example inserts an HTML INPUT element and some replacement text into the output document if there is exactly one replacement data item named "responder-dn" in the replacement data.

## <form:menu>

The form:menu element is used to generate an HTML SELECT element with one or more OPTION element children. The first OPTION element child will be marked as selected.

### Attributes

**name:** Specifies the name of the replacement data item. If the named item appears in the replacement data, then an HTML SELECT element is created in the output document. An HTML OPTION element is created as a child of the SELECT element for each instance of the replacement data item in the replacement data.

### Example

```
<form:menu name="responder-dn"/>
```

The above example will result in HTML elements similar to the following:

```
<SELECT name="responder-dn">  
  <OPTION selected>\PERIN-TAO\big-org\php</OPTION>  
  <OPTION>\PERIN-TAO\big-org\carol</OPTION>  
</SELECT>
```

## Appendix D - <mail> Element Reference

The <mail> element and its content are described in detail in this section. If no attributes are listed for an element, then that element has no attributes defined.

### <mail>

The <mail> element and its content describe the data necessary to construct an SMTP message.

#### <mail> attributes

**src-dn:** Contains the DN value of the eDirectory object that is triggering the e-mail . Required if the object's data is to be modified via the Publisher channel's Web server in response to the e-mail

### <to>

The <to> element is a child of the <mail> element. One or more <to> elements contain the e-mail addresses of the primary recipients of the SMTP message. At least one <to> element is required. Each <to> element must contain only a single e-mail address.

### <cc>

The <cc> element is a child of the <mail> element. Zero or more <cc> elements contain the e-mail addresses of the CC recipients of the SMTP message. No <cc> element is required. Each <cc> element must contain only a single e-mail address.

### <bcc>

The <bcc> element is a child of the <mail> element. Zero or more <bcc> elements contain the e-mail addresses of BCC recipients of the SMTP message. No <bcc> element is required. Each <bcc> element must contain only a single e-mail address.

### <from>

The <from> element is a child of the <mail> element. The <from> element contains the e-mail address of the sender of the e-mail . The <from> element is not required. If the <from> element is not present, then the default from address supplied as part of the Manual Task Service Driver parameters is used.

### <reply-to>

The <reply-to> element is a child of the <mail> element. The <reply-to> element contains the e-mail address of the entity to which replies to the SMTP message will be addressed. The <reply-to> element is not required.

### <subject>

The <subject> element is a child of the <mail> element. Its string content is used to set the SMTP subject field. The <subject> element is not required but is recommended, for obvious reasons.

## <message>

The <message> element is a child of the <mail> element. Its content is used to construct a message body for the SMTP message. At least one <message> element is required. Multiple <message> elements can be supplied when constructed an SMTP message with alternative representations of the message body (such as plain text and HTML, or English and another language).

### <message> attributes

**mime-type:** Optionally specifies the MIME type of the message body constructed by the <message> element (such as text/plain or text/html). If the mime-type attribute is not present, the driver will attempt to automatically discover the mime-type.

E-mail clients can use the MIME type when an SMTP message has alternative representations in order to choose the best representation to display.

**language:** Optionally specifies the language of the message body constructed by the <message> element. The value should follow the SMTP specification. If the language attribute is not present no default is supplied.

E-mail clients can use the language specification when an SMTP message has alternative representations in order to choose the best representation to display.

## <stylesheet>

The <stylesheet> element is a child of the <message> element. The content of the <stylesheet> element is the name of an XSLT style sheet used to construct the message body. If the <stylesheet> element is not present then process\_template.xsl is used as the style sheet.

## <template>

The <template> element is a child of the <message> element. The content of the <template> element is the name of an XML document used to construct the message body. If the <template> element is not present, then the replacement data document is processed by the message style sheet to construct the message body.

## <filename>

The <filename> element is a child of the <attachment> element. The content of the <filename> element is a filename. The filename value is used to assign a filename to a constructed attachment.

## <replacement-data>

The <replacement-data> element is a child of the <message> element. Its content is used either as a parameter to the style sheet processing the message template, or in the absence of a template, it is processed directly by the message style sheet. The contents of the <replacement-data> element are described fully elsewhere in Appendixes A and B.

## <resource>

The <resource> element is a child of the <message> element. Its content is treated as the name of a file to be incorporated into the SMTP message a resource for the message body. For example, a .css style sheet for an HTML message body could be supplied as a resource.

### <resource> attributes

**cid:** Specifies the content ID used to refer to the resource in URLs in the message body. For example, if a .css style sheet is the resource, then the cid value might be css-1. In the HTML message body the following element can be used to refer to the .css style sheet:  
<link href="cid:css-1" rel="style sheet" type="text/css">.

## <attachment>

The <attachment> element is a child of the <mail> element. It can have the same content as <message>, or it may have a filename as content. Zero or more <attachment> elements can appear as children of the <mail> element.

### <attachment> attributes

**mime-type:** Optionally specifies the MIME type of the attachment. If the mime-type attribute is not present, the driver will attempt to automatically discover the MIME type.

**language:** Optionally specifies the language of the attachment. If the language attribute is not present, no default is supplied.

## Appendix E – Data Flow

This section gives a step-by-step examination of the data flow in an example situation when hiring a new employee causes an e-mail message to be sent to the employee's manager. The e-mail message requests that the manager use a URL in the message to enter a room number value for the employee.

The configuration of the Manual Task Service Driver is as follows for the example scenario:

### Subscriber Channel Configuration

#### Filter

Class: User

Attributes:

- Given Name
- manager
- Surname

#### Policies

Create policy:

Requires Given Name, manager, and Surname attributes.

Command Transformation policy

Converts <add> into <mail> element.

### Publisher Channel Configuration

#### Filter

Class: User

Attributes:

- roomNumber

#### Policies

None

### Step-by-Step Examination

In the following list, the most important data items that flow through the process are responder-dn and association. The responder-dn item is used to authenticate the user entering data through the Web server. The association item identifies the eDirectory object whose data is to be changed.

- 1) The company hires a new employee. The new employee's data is entered into the company's Human Resource (HR) system.
- 2) The DirXML driver for the HR system creates a new User object in eDirectory. User attributes include Given Name, Surname, and manager.

- 3) The following <add> event for the new User object is submitted to the Manual Task Service Driver Subscriber channel:

```
<nds dtdversion="1.1" ndsversion="8.6">
  <input>
    <add class-name="User" src-dn="\PERIN-TAO\novell\Provo\Joe" src-entry-id="281002"
timestamp="1023314433#2">
      <add-attr attr-name="Surname">
        <value type="string">the Intern</value>
      <add-attr>
        <add-attr attr-name="Given Name">
          <value type="string">Joe</value>
        <add-attr>
          <add-attr attr-name="manager">
            <value type="dn">\PERIN-TAO\novell\Provo\phb</value>
          <add-attr>
        </add>
      </input>
    </nds>
```

- The Subscriber Command Transformation policy uses the manager DN value to issue a query to eDirectory for the manager's e-mail address and the manager's assistant's DN.
- If the manager has an assistant the Subscriber Command Transformation issues a query to eDirectory for the assistant's e-mail address.
- The Subscriber Command Transformation constructs a <mail> element and replaces the <add> command element with the <mail> element. Replacement data items are in **bold**.

```
<nds dtdversion="1.1" ndsversion="8.6">
  <input>
    <mail src-dn="\PERIN-TAO\novell\Provo\Joe">
      <to>phb@company.com</to>
      <cc>carol@company.com</cc>
      <bcc>HR@company.com</bcc>
      <reply-to>HR@company.com</reply-to>
      <subject>Room Assignment Needed for: Joe the Intern</subject>
      <message mime-type="text/html">
        <stylesheet>process_template.xsl</stylesheet>
        <template>html_msg_template.xml</template>
        <replacement-data>
          <item name="manager">JStanley</item>
          <item name="given-name">Joe</item>
          <item name="surname">the Intern</item>
          <url-data>
            <item name="file">process_template.xsl</item>
            <url-query>
              <item name="template">form_template.xml</item>
              <item name="responder-dn" protect="yes">\PERIN-TAO\novell\Provo\phb</item>
              <item name="responder-dn" protect="yes">\PERIN-TAO\novell\Provo\carol</item>
              <item name="subject-name">Joe the Intern</item>
            </url-query>
          </url-data>
        </replacement-data>
        <resource cid="css-1">novdocmain.css</resource>
      </message>
    </mail>
  </input>
</nds>
```

- The Manual Task Service Driver Subscriber receives the <mail> element from DirXML.
- The Subscriber generates an association value because the <mail> element has a *src-dn* attribute.

- f) The Subscriber constructs a replacement data document from the data in the <mail> element for use in constructing the e-mail message. The URL has various data items in the query portion (that portion of the URL that follows the '?' character and is in **bold**). The Publisher channel Web server will use these data items when the URL is submitted to the Web server as an HTTP GET request.

```

<replacement-data>
  <item name="manager">JStanley</item>
  <item name="given-name">Joe</item>
  <item name="surname">the Intern</item>
  <item name="template">form_template.xml</item>
  <item name="responder-dn">\PERIN-TAO\novell\Provo\phb</item>
  <item name="responder-dn">\PERIN-TAO\novell\Provo\carol</item>
  <item name="subject-name">Joe the Intern</item>
  <item name="association">1671b2:ee4246a561:-7fff:192.168.0.1</item>
  <item name="url-base">https://192.168.0.1:8180</item>
  <item name="url-file">process_template.xsl</item>
  <item name="protected-data">
r00ABXNyAb1qYXZheC5jcnldG8uU2VhbGVkT2JqZWN0PjY9psO3VHACAARbAA
11bmNvZGVkUGFyYW1zdAACW0JbABB1bmNyeXB0ZWRDb250ZW50cQB+AAFMAAlw
YXJhbXNbbGd0ABJMamF2YS9sYW5nLlN0cm1uZztMAAdzZW50cXB+AAJ4cH
VyAAJbQqzzF/gGCFTgAgAAeHAAAAAPMA0ECEIBRohGPjxEAgEKdXEafgAEAAAA
uMSFqzHXwtMx8DkRCzkK1O46sEzlu51o3MDvHn+3+fE6SphHr3HgJli4Jp3rUk
H7y6dXvcu7iq21Vs+9o6iZVz1jTtjX/jjRrVZ1R5JouRNhk8JHFZ8FhgsmiIAH
/Fs61k4WmyEcmYfWmfqfBVeThr3Avwcm6ranS5Mm2U5i9Z/DBR13pIAobMpwY
kMaz4+G9e6oovBsiPdp6jSPzbFxcgALi2AMBh4hf9jnx7zOU9Uvd9qXtaE2rR0
AANQQkV0ABBQkVXaXRoTUQ1QW5kREVT</item>
  <item name="url-query">template=form_template.xml&responder-dn=%5CPERIN-
TAO%5Cnovell%5Cprovo%5Cphb&responder-dn=%5CPERIN-
TAO%5Cnovell%5Cprovo%5Ccarol&subject-
name=Joe+the+Intern&association=1671b2%3Aee4246a561%3A-
7fff%3A192.168.0.1&protected-
data=r00ABXNyAb1qYXZheC5jcnldG8uU2VhbGVkT2JqZWN0PjY9psO3VHACAARbAA11bmNvZGVkUGFyYW1zdAAC
W0JbABB1bmNyeXB0ZWRDb250ZW50cQB%2BAAFMAAlwYXJhbXNbbGd0ABJMamF2YS9sYW5nLlN0cm1uZztMAAdzZW50cXB%2BAAJ4cH
VYAAJbQqzzF%2FgGCFTgAgAAeHAAAAAPMA0ECEIBRohGPjxEAgEKdXEafgAEAAAAuMSFqzHX
wtMx8DkRCzkK1O46sEzlu51o3MDvHn%2B3%2BF6SphHr3HgJli4Jp3rUkH7y6dXvcu7iq21Vs%2B9o6iZVz1jTtj
X%2FjjRrVZ1R5JouRNhk8JHFZ8FhgsmiIAH%2FFS61k4WmyEcmYfWmfqfBVeThr3Avwcm6ranS5Mm2U5i9Z%2FDB
R13pIAobMpwYkMaz4%2BG9e6oovBsiPdp6jSPzbFxcgALi2AMBh4hf9jnx7zOU9Uvd9qXtaE2rR0AANQQkV0ABBQ
kVXaXRoTUQ1QW5kREVT</item>
  <item name="url">
https://192.168.0.1:8180/process_template.xsl?template=form_template.xml&responder-
dn=%5CPERIN-TAO%5Cnovell%5Cprovo%5Cphb&responder-dn=%5CPERIN-
TAO%5Cnovell%5Cprovo%5Ccarol&subject-
name=Joe+the+Intern&association=1671b2%3Aee4246a561%3A-
7fff%3A192.168.0.1&protected-
data=r00ABXNyAb1qYXZheC5jcnldG8uU2VhbGVkT2JqZWN0PjY9psO3VHACAARbAA11bmNvZGVkUGFyYW1zdAAC
W0JbABB1bmNyeXB0ZWRDb250ZW50cQB%2BAAFMAAlwYXJhbXNbbGd0ABJMamF2YS9sYW5nLlN0cm1uZztMAAdzZW50cXB%2BAAJ4cH
VYAAJbQqzzF%2FgGCFTgAgAAeHAAAAAPMA0ECEIBRohGPjxEAgEKdXEafgAEAAAAuMSFqzHX
wtMx8DkRCzkK1O46sEzlu51o3MDvHn%2B3%2BF6SphHr3HgJli4Jp3rUkH7y6dXvcu7iq21Vs%2B9o6iZVz1jTtj
X%2FjjRrVZ1R5JouRNhk8JHFZ8FhgsmiIAH%2FFS61k4WmyEcmYfWmfqfBVeThr3Avwcm6ranS5Mm2U5i9Z%2FDB
R13pIAobMpwYkMaz4%2BG9e6oovBsiPdp6jSPzbFxcgALi2AMBh4hf9jnx7zOU9Uvd9qXtaE2rR0AANQQkV0ABBQ
kVXaXRoTUQ1QW5kREVT</item>
</replacement-data>

```

- g) The Subscriber processes `html_msg_template.xml` with `process_template.xml`. The replacement data document is passed as a parameter to the style sheet. The `html_msg_template.xml` document follows. Note the replacement tokens in bold. The replacement tokens will be replaced by the value of the corresponding `<item>` elements in the replacement data document.

```
<html xmlns:form="http://www.novell.com/dirxml/workflow/form">
<head>
</head>
<body>
<link href="cid:css-1" rel="style sheet" type="text/css"/>
<p>
Dear $manager$,
</p>
<p>
This message is to inform you that your new employee <b>$given-name$ $surname$</b> has
been hired.
</p>
<p>
Please assign a room number for this individual. Click <a href="$url$">Here</a> to do
this.
</p>
<p>
Thank you,<br/>
HR<br/>
HR Department
</p>
</body>
</html>
```

- The generated e-mail document follows. Note the replacement tokens have been replaced with the values of the corresponding `<item>` elements from the replacement data document.

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<link href="cid:css-1" rel="style sheet" type="text/css">
<p>
Dear J Stanley,
</p>
<p>
This message is to inform you that your new employee <b>Joe the Intern</b> has been
hired.
</p>
<p>
Please assign a room number for this individual. Click <a
href="https://192.168.0.1:8180/process_template.xml?template=form_template.xml&responder-
dn=%5CPERIN-TAO%5Cnovell%5CProvo%5Cphb&responder-dn=%5CPERIN-
TAO%5Cnovell%5CProvo%5Ccarol&subject-
name=Joe+the+Intern&association=45f0e3%3Aee45e07709%3A-7fff%3A192.168.0.1&protected-
data=r00ABXNyAB1qYXZheC5jcnlwdG8uU2VhbGVkT2JqZWN0PjY9psO3VHACAARbAA1lbmNvZGVkUGFyYV1zdAAC
W0JbABB1bmNyeXB0ZWRDb250ZW50cQB%2BAAFMAAlwYXJhbXNbbGd0ABJMamF2YS9sYW5nL1N0cm1uZztMAAdzZWF
sQWxncQB%2BAAJ4cHVyAAJbQqzzF%2FgGCFtgAgAAeHAAAAAPMA0ECir9Z1iG%2B03BAgEKdXEafgAEAAAAuMU%2F
SoFRkebhv2d5SgalF91ttjRY51yyW5%2B%2FFIfOuDDyikYiDbOJb6607S0dPHJQzeVgu6ptIvGqaEQOEjBjDkY%2
Bi4VoVjUSXS3a8fiXB8moMgPtLJ%2FGyE8Qiwbt4xbkQy48i02k99F2vGmlenRpSP6dD31kZ13dpJ0mGgq2yL%2Fe
FaynKyqnjkhLMexcqD8W1Vooar11k2RPk5vDYvC8o2bn22OKKbOnSRM5Y1PS0iWzxo0JVcnVvYt0AANQQkV0ABBQQ
kVXaXRoTUQ1QW5kREVT">Here</a> to do this.
</p>
<p>
Thank you,<br>
HR<br>
HR Department
</p>
</body>
</html>
```

- h) The SMTP e-mail message is sent to the manager and to the manager's assistant.

- i) The Subscriber returns an XML document containing a <status> element and an <add-association> element to DirXML.
- 4) The manager opens the e-mail message and clicks the “Click here” link.
- 5) The manager’s Web browser submits the URL to the Publisher channel Web server as an HTTP GET request.
  - a) The Web server constructs the following replacement data document. Most of the data items come from the query portion of the URL. The exceptions are the automatically generated items url and url-base.

```

<replacement-data>
  <item name="association">45f0e3:ee45e07709:-7fff:192.168.0.1</item>
  <item name="protected-
data">rO0ABXNyABlqYXZheC5jcnlwdG8uU2VhbGVkT2JqZWNOPljY9psO3VHACAARbAA1lbmNvZGVkUGFyYW1zdAA
CW0JbABBlbmNyeXB0ZWRDb250ZW50cQB+AAFMAAlwYXJhbXNBbGd0ABJMamF2YS9sYW5nLlN0cm1uZztMAadzZWFs
QWxncQB+AAJ4cHVyAAJbQqzzF/gGCFTgAgAAeHAAAAAPMA0ECIr9Z1iG+O3BAgEKdXEafgAEAAAAuMU/SoFRkebvH
2d5SgalF91ttjRY5lyyW5+/FifOuDdYikYiDbOJb6607S0dPHjQzeVgu6ptIvGqaEQOEjBjDkY+i4VoVjUSXS3a8f
iXB8moMdPtLJ/GyE8Qiwbt4xbkQy48i02k99F2vGmlenRpsP6dD31kZ13dpJ0mGgq2yL/eFaynKyqnjkHLMexcqD8
WlVooaR1lk2RPk5vDYvC8o2bn22OKKbOnSRM5YlPS0iWzxo0JVcnVVyt0AANQQkV0ABBQQkVXaXRoTUQ1QW5kREVT
</item>
  <item name="template">form_template.xml</item>
  <item name="responder-dn">\PERIN-TAO\novell\Provo\phb</item>
  <item name="responder-dn">\PERIN-TAO\novell\Provo\carol</item>
  <item name="subject-name">Joe the Intern</item>
  <item name="url-base">https://192.168.0.1:8180</item>
  <item name="url">https://192.168.0.1:8180</item>
</replacement-data>

```

The Web server processes the form\_templates.xml document with the process\_template.xml style sheet. Replacement tokens and action elements are in **bold**. Note that various data items are placed in hidden INPUT elements so that the data items are passed to the Web server as part of the HTML POST data.

In addition, there is a \$query:roomNumber\$ replacement token, which will retrieve the current value of the employee's roomNumber attribute (if any).

```
<html xmlns:form="http://www.novell.com/dirxml/workflow/form">
  <head>
    <title>Enter room number for $subject-name$</title>
  </head>
  <body>
    <link href="novdocmain.css" rel="style sheet" type="text/css"/>
    <br/><br/><br/><br/>
    <form class="myform" METHOD="POST" ACTION="$url-base$/process_template.xml">
      <table cellpadding="5" cellspacing="10" border="1" align="center">
        <tr><td>
          <input TYPE="hidden" name="template" value="post_form.xml"/>
          <input TYPE="hidden" name="subject-name" value="$subject-name$" />
          <input TYPE="hidden" name="association" value="$association$" />
          <input TYPE="hidden" name="response-style sheet" value="process_template.xml"/>
          <input TYPE="hidden" name="response-template" value="post_response.xml"/>
          <input TYPE="hidden" name="auth-style sheet" value="process_template.xml"/>
          <input TYPE="hidden" name="auth-template" value="auth_response.xml"/>
          <input TYPE="hidden" name="protected-data" value="$protected-data$" />
          <form:if-single-item name="responder-dn">
            You are:<br/>
            <input TYPE="hidden" name="responder-dn" value="$responder-dn$" />
            $responder-dn$
          </form:if-single-item>
          <form:if-multiple-items name="responder-dn">
            Indicate your identity:<br/>
            <form:menu name="responder-dn"/>
          </form:if-multiple-items>
        </td></tr>
        <tr><td>
          Enter your password: <br/><input name="password" TYPE="password" SIZE="20"
          MAXLENGTH="40" />
        </td></tr>
        <tr><td>
          Enter room number for $subject-name$:<br/>
          <input TYPE="text" NAME="room-number" SIZE="20" MAXLENGTH="20"
          value="$query:roomNumber$" />
        </td></tr>
        <tr><td>
          <input TYPE="submit" value="Submit" /> <input TYPE="reset" value="Clear" />
        </td></tr>
      </table>
    </form>
  </body>
</html>
```

The following HTML page results:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Enter room number for Joe the Intern</title>
</head>
<body>
<link href="novdocmain.css" rel="style sheet" type="text/css">
<br><br><br><br>
<form class="myform" METHOD="POST"
ACTION="https://192.168.0.1:8180/process_template.xml">
<table cellpadding="5" cellspacing="10" border="1" align="center">
<tr>
<td>
<input TYPE="hidden" name="template" value="post_form.xml">
<input TYPE="hidden" name="subject-name" value="Joe the Intern">
<input TYPE="hidden" name="association" value="45f0e3:ee45e07709:-7fff:192.168.0.1">
<input TYPE="hidden" name="response-style sheet" value="process_template.xml">
<input TYPE="hidden" name="response-template" value="post_response.xml">
<input TYPE="hidden" name="auth-style sheet" value="process_template.xml">
<input TYPE="hidden" name="auth-template" value="auth_response.xml">
<input TYPE="hidden" name="protected-data"
value="r00ABXNyABlqYXZheC5jcnlwdG8uU2VhbGVkT2JqZWNPjY9psO3VHACAARbAA11bmNvZGVkUGFyYw1zdA
ACW0JbABB1bmNyeXB0ZWRDb250ZW50cQB+AAFMAAlwYXJhbXNBbGd0ABJMamF2YS9sYW5nLn0cmcluZztMAAdzZWF
sQWxncQB+AAJ4cHVyAAJbQqzzF/gGCFtgAgAAeHAAAAAPMA0ECIr9Z1iG+O3BAGeKdXEafgAEAAAAuMU/SoFRkebv
h2d5SsqalF91ttjRY51yyW5+/FIfoUddYikYiDb0Jb6607S0dPHjQzeVgu6ptIvGqaEQOEjBjDkY+i4VoVjUSXS3a8
fiXB8moMdPtLJ/GyE8Qiwbt4xbkQy48i02k99F2vGmlenRpSP6dD3lkZl3dpJ0mGgq2yL/eFaynKyqnjkhLMexcqD
8WlVooaR11k2Rpk5vDYvc8o2bn22OKKbOnSRM5YlPS0iWzxo0JvCnVVyt0AANQQkV0ABBQkVXaXRoTUQ1QW5kREV
T">
Indicate your identity:<br>
<SELECT name="responder-dn">
<OPTION selected>\PERIN-TAO\novell\Provo\phb</OPTION>
<OPTION>\PERIN-TAO\novell\Provo\carol</OPTION>
</SELECT>
</td>
</tr>
<tr>
<td>
Enter your password: <br>
<input name="password" TYPE="password" SIZE="20" MAXLENGTH="40">
</td>
</tr>
<tr>
<td>
Enter room number for Joe the Intern:<br>
<input TYPE="text" NAME="room-number" SIZE="20" MAXLENGTH="20" value="">
</td>
</tr>
<tr>
<td>
<input TYPE="submit" value="Submit"> <input TYPE="reset" value="Clear">
</td>
</tr>
</table>
</form>
</body>
</html>
```

- b) The manager selects his eDirectory DN from the Web page menu, enters his password, enters the room number for the new employee, and clicks the Submit.
- c) The Web browser submits an HTTP POST request to the Web server.
- d) The Web server constructs the following replacement data document from the POST data. Note the data that was in the various hidden <INPUT> elements. The data entered by the manager in the form is in **bold**.

```

<replacement-data>
  <item name="room-number">cubicle 1234</item>
  <item name="template">post_form.xml</item>
  <item name="response-template">post_response.xml</item>
  <item name="auth-template">auth_response.xml</item>
  <item name="association">45f0e3:ee45e07709:-7fff:192.168.0.1</item>
  <item name="password" is-sensitive="true"><!--content suppressed --></item>
  <item name="protected-
data">r00ABXNyABlqYXZheC5jcnlwdG8uU2VhbGVkT2JqZWN0PjY9psO3VHACAARbAA11bmNvZGVkUGFyYW1zdAA
CW0JbABBlbmNyeXB0ZWRDb250ZW50cQB+AAFMAAlwYXJhbXNBbGd0ABJMamF2YS9sYW5nL1N0cm1uZztMAAdzZWFs
QWxncQB+AAJ4cHVyAAJbQqzzF/gGCFTgAgAAeHAAAAAPMA0ECIr9Z1iG+O3BAGeKdXEafgAEAAAAuMU/SofRkebvH
2d5Sqa1F91tttjRY5lyyW5+/FifOuDDyikYiDbOJb6607S0dPHjQzeVgu6ptIvGqaEQOEjBjDkY+i4VoVjUSXS3a8f
iXB8moMdPtLJ/GyE8Qiwbt4xbkQy48i02k99F2vGmlenRpSP6dD31kZl3dpJ0mGgq2yL/eFaynKyqnjkHLMexcqD8
WlVooaR1lk2RPk5vDYvC8o2bn22OKKbOnSRM5Y1PS0iWzxo0JVcnVVyt0AANQQkV0ABBQQkVXaXR0TUQ1QW5kREVT
</item>
  <item name="responder-dn">\PERIN-TAO\novell\Provo\phb</item>
  <item name="auth-style sheet">process_template.xsl</item>
  <item name="response-style sheet">process_template.xsl</item>
  <item name="subject-name">Joe the Intern</item>
  <item name="url-base">https://192.168.0.1:8180</item>
  <item name="url">https://192.168.0.1:8180</item>
</replacement-data>

```

- e) The Web server verifies that the value of item responder-dn matches a responder-dn value contained in the protected data. If the value does not match, the Web server aborts the request. If the value does match, processing continues.
- f) The Web server submits a <check-object-password> XDS request to DirXML on the Publisher channel to authenticate the user submitting the HTTP POST request.

```

<nds dtdversion="1.0" ndsversion="8.6">
  <source>
    <product build="20020606_0824" instance="Workflow Request Service Driver"
version="1.1a">DirXML Workflow Request Service Driver</product>
    <contact>Novell, Inc.</contact>
  </source>
  <input>
    <check-object-password dest-dn="\PERIN-TAO\novell\Provo\phb" event-id="chkpwd">
      <password><!-- content suppressed --></password>
    </check-object-password>
  </input>
</nds>

```

- g) DirXML returns <status level="success">. If DirXML returns other than success, then the templates specified by the data item auth\_template and the style sheet specified by the data item auth\_stylesheet are used to construct a Web page that is returned as the result of the POST.

- h) The Web server processes the post\_form.xml template with the process\_template.xsl style sheet to generate an XDS document. Replacement tokens are in **bold**.

```
<nds>
  <input>
    <modify class-name="User" src-dn="not-applicable" event-id="wfmod">
      <association>$association$</association>
      <modify-attr attr-name="roomNumber">
        <remove-all-values/>
        <add-value>
          <value>$room-number$</value>
        </add-value>
      </modify-attr>
    </modify>
  </input>
</nds>
```

- i) The Publisher submits the created XDS document to DirXML.

```
<nds>
  <input>
    <modify class-name="User" src-dn="not-applicable" event-id="wfmod">
      <association>45f0e3:ee45e07709:-7fff:192.168.0.1</association>
      <modify-attr attr-name="roomNumber">
        <remove-all-values/>
        <add-value>
          <value>cubicle 1234</value>
        </add-value>
      </modify-attr>
    </modify>
  </input>
</nds>
```

- j) DirXML returns a result document

```
<nds dtdversion="1.1" ndsversion="8.6">
  <source>
    <product version="1.1a">DirXML</product>
    <contact>Novell, Inc.</contact>
  </source>
  <output>
    <status event-id="wfmod" level="success"></status>
  </output>
</nds>
```

- k) The Web server adds the replacement data item post-status (and possibly the replacement data item post-status-message) to the replacement data document. The added data item is in **bold**:

```
<replacement-data>
  <item name="room-number">cubicle 1234</item>
  <item name="template">post_form.xml</item>
  <item name="response-template">post_response.xml</item>
  <item name="auth-template">auth_response.xml</item>
  <item name="association">45f0e3:ee45e07709:-7fff:192.168.0.1</item>
  <item name="password" is-sensitive="true"><!--content suppressed --></item>
  <item name="protected-
data">r00ABXNyABlqYXZheC5jcnldwG8uU2VhbGVkT2JqZWNPpJy9psO3VHACAARbAA11bmNvZGVkUGFyYW1zdAA
CW0JbABBlbmNyeXB0ZWRDb250ZW50cQB+AAFMAAlwYXJhbXNBbGd0ABJMamF2YS9sYW5nLlN0cmduZztMAAdzZWFS
QWxncQB+AAJ4cHVyAAJbQqzzF/gGCFTgAgAAeHAAAAAPMA0ECIr9Z1iG+O3BAGeKdXEafgAEAAAAuMU/SoFRkebvH
2d5Sqa1F91ttjRY5lyyW5+/FifOuDdYikYiDb0Jb6607S0dPHjQzeVgu6ptIvGqaEQOEjBjDkY+i4VoVjUSXS3a8f
iXB8moMdPtLJ/GyE8QiwBT4xbkQy48i02k99F2vGmlenRpSP6dD31kZl3dpJ0mGgq2yL/eFaynKyqnjkHLMexccqD8
WlVooar1lk2Rpk5vDYvC8o2bn22OKKbOnSRM5YlPS0iWzxo0JVcnVVyt0AANQQkV0ABBQQkVXaXRoTUQ1QW5kreVT
</item>
  <item name="responder-dn">\PERIN-TAO\novell\Provo\phb</item>
  <item name="auth-style sheet">process_template.xsl</item>
  <item name="response-style sheet">process_template.xsl</item>
  <item name="subject-name">Joe the Intern</item>
  <item name="url-base">https://192.168.0.1:8180</item>
  <item name="url">https://192.168.0.1:8180</item>
  <status event-id="" level="success"></status>
  <item name="post-status">success</item>
</replacement-data>
```

- l) The Web server processes the post\_response.xml template with the process\_template.xsl style sheet. Replacement tokens and action elements are in **bold**.

```
<htm xmlns:form="http://www.novell.com/dirxml/workflow/form">
  <head>
    <title>Result of post for $subject-name$</title>
  </head>
  <body>
    <link href="novdocmain.css" rel="style sheet" type="text/css"/>
    <br/><br/><br/><br/>
    <table class="formtable" cellpadding="5" cellspacing="20" border="1" align="center">
      <tr>
        <td>
          DirXML reported status = $post-status$
        </td>
      </tr>
      <form:if-item-exists name="post-status-message">
        <tr>
          <td>
            Status message was: $post-status-message$
          </td>
        </tr>
      </form:if-item-exists>
    </table>
  </body>
</html>
```

- m) The resulting Web page is returned as the result of the HTTP POST. Note that the second row of the table is not present because the post-status-message referred to by the <form:if-item-exists> element is not present in the replacement data document.

```
<html>
  <head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Result of post for Joe the Intern</title>
  </head>
  <body>
    <link href="novdocmain.css" rel="style sheet" type="text/css">
    <br><br><br><br>
    <table class="formtable" cellpadding="5" cellspacing="20" border="1" align="center">
      <tr>
        <td>
          DirXML reported status = success
        </td>
      </tr>
    </table>
  </body>
</html>
```

## Appendix F – Custom Element Handlers for the Subscriber Channel

The Workflow driver provides an extension mechanism for sending user notifications using methods other than the Simplified Mail Transport Protocol (SMTP). For example, a customer might have a need to send notifications using the Messaging Application Programming Interface (MAPI) rather than using SMTP.

To use a mechanism other than SMTP for sending notifications it is necessary to write a Java class that handles a custom XML element that is submitted on the Workflow driver's Subscriber channel.

The Java custom element handler must implement the Java interface `com.novell.nds.dirxml.driver.workflow.CommandHandler`. The name of the custom element class is specified in the Additional Handlers item found in the Subscriber configuration parameters.

When the workflow driver Subscriber encounters a command element it looks in its table of handlers. When it finds a handler that reports that it handles the command element the command element is passed to the handler. The handler then performs any processing required.

There are two built-in command element handlers in the Workflow driver: a handler for `<mail>` elements and a handler for `<add>` elements.

The custom command element definition is up to the author of the custom handler. A reasonable place to start in designing the custom command element is the design of the `<mail>` element.

The custom elements are created by policies on the subscriber channel in the same fashion that the `<mail>` element is created.

The documentation for `com.novell.nds.dirxml.driver.workflow.CommandHandler` and the documentation for many utility and support classes are found in the javadocs that ship with the Workflow driver. The javadocs may be found in the file named `workflow_driver_docs.zip` in the distribution image.

### Constructing URLs for use with the Publisher Channel Web Server

To securely use the Workflow driver's Publisher channel web server it is necessary to use utility classes to construct the URL that is to be included with a notification message. The `com.novell.nds.dirxml.driver.workflow.URLData` is designed for this task.

The sample code found in `SampleCommandHandler.java` illustrates this process.

### Constructing Message Documents using Stylesheets and Template Documents

It is convenient to use the same method to construct documents that the SMTP handler uses, namely, a combination of stylesheets, template documents, and replacement data. In order to accomplish this it is necessary to obtain the stylesheets and template documents, and to invoke the stylesheet processor programmatically.

The sample code found in `SampleCommandHandler.java` illustrates this process.

## SampleCommandHandler.java

Source code for a sample custom command handler is included with the Workflow driver distribution. The source code may be found in the file `workflow_driver_docs.zip` which is found in the distribution image.

The handler is implemented in the class `com.novell.nds.dirxml.driver.workflow.samples.SampleCommandHandler`.

The sample handler simply generates a document using stylesheets and templates and writes the resulting document in a file.

### Compiling the SampleCommandHandler Class

To compile the `SampleCommandHandler` class you can use any Java 2 compiler. It is necessary to place `nxsl.jar`, `dirxml.jar`, `collections.jar`, and `WorkflowRequestServiceBase.jar` in the java compiler classpath.

### Trying the SampleCommandHandler Class

Start by importing the Room Number sample configuration for the Workflow driver.

Compile the `SampleCommandHandler` class and place the resulting class file in a `.jar` file. Place the `.jar` file in the `DirXML.jar` file directory appropriate to the platform on which you are running the driver.

Add the following XML element under the `<subscriber-options>` element found in the Driver Parameters XML section of the driver properties:

```
<output-path display-name="Sample Output Path"></output-path>
```

Edit the Driver Parameters. In the item labeled *Sample Output Path* place a path to a directory in which the `SampleCommandHandler` will write its created documents. In the item labeled *Additional handlers* add the string `com.novell.nds.dirxml.driver.workflow.samples.SampleCommandHandler`.

Replace the Subscriber channel command transformation policy with `CommandXform.xml` which may be found in the same directory as the `SampleCommandHandler.java` file.

Create a User object and add a manager reference to the User object. If the manager has an e-mail address value then a `<sample>` command element will be sent to the Subscriber and the `SampleCommandHandler` will write a file in the location you specified above.

# Appendix G – Custom Servlets for the Publisher Channel

The Workflow driver provides an extension mechanism through which additional functionality can be added to the Publisher channel web server. Custom servlets can be loaded by the Publisher by specifying the name of the servlet class(es) in the Driver configuration item labeled *Additional servlets*.

## Using the Publisher Channel

If a custom servlet needs to submit data to DirXML the servlet will need to use the Workflow driver's Publisher channel. The classes `com.novell.nds.dirxml.driver.workflow.ServletRegistrar` and `com.novell.nds.dirxml.driver.workflow.PublisherData` are supplied to facilitate this. The sample code found in `SampleServlet.java` illustrates this process.

## Authentication

A custom servlet must authenticate users that are submitting information. The sample code found in `SampleServlet.java` illustrates this process. Note, however, the type of authentication performed using the `<check-object-password>` element does not check NDS rights. Changes submitted on the Publisher channel will be allowed if the Driver object has rights to perform the changes regardless of whether the user submitting the changes has rights or not.

If using a URL generated by a command handler on the Subscriber channel it is necessary to use the `com.novell.nds.dirxml.driver.workflow.URLData` class to validate the URL to ensure that the `responder-dn` data item has not been tampered with. See the javadocs for information on accomplishing this.

## SampleServlet.java

Source code for a sample servlet is included with the Workflow driver distribution. The source code may be found in the file `workflow_driver_docs.zip` which is found in the distribution image.

The servlet is implemented in the class `com.novell.nds.dirxml.driver.workflow.samples.SampleServlet`.

The sample servlet accepts an HTTP GET request for any resource ending in `.sample`. The query string of the HTTP URL must contain a `dest-dn` item, an `attr-name` item, and a `value` item.

The servlet authenticates the user and then submits a modify request to DirXML via the Workflow driver's Publisher channel.

## Compiling the SampleServlet Class

To compile the `SampleServlet` class you can use any Java 2 compiler. It is necessary to place `nxsl.jar`, `dirxml.jar`, `collections.jar`, and `WorkflowRequestServiceBase.jar` in the java compiler classpath.

## Trying the SampleServlet Class

Start by importing the Room Number sample configuration for the Workflow driver.

Compile the `SampleServlet` class and place the resulting class file in a `.jar` file. Place the `.jar` file in the DirXML `.jar` file directory appropriate to the platform on which you are running the driver.

Edit the Driver Parameters. In the item labeled *Additional servlets* add the string `com.novell.nds.dirxml.driver.workflow.samples.SampleServlet`.

Add Telephone Number to the Publisher channel filter.

Submit the following URL in a browser (assuming the browser is running on the same machine as the Workflow driver):

<https://localhost:8180/1.sample?dest-dn=username.container&attr-name=Telephone%20Number&value=555-1212>

Replace *username.container* with the DN of a user in your tree.