



TECHNICAL REFERENCE GUIDE : USER INTERFACES

Technical Reference Guide: User Interfaces

For ViewDS Release 7.4 Patch 4

August 2016

This publication is copyright. Other than for the purposes of and subject to the conditions prescribed under the Copyright Act, no part of it may in any form or by any means (electronic, mechanical, microcopying, photocopying, recording or otherwise) be reproduced, stored in a retrieval system or transmitted without prior written permission. Inquiries should be addressed to the publishers.

The contents of this publication are subject to change without notice. All efforts have been made to ensure the accuracy of this publication. Notwithstanding, eNitiatives.com Pty. Ltd. does not assume responsibility for any errors nor for any consequences arising from any errors in this publication.

The software and/or databases described in this document are furnished under a licence agreement. The software and/or databases may be used or copied only in accordance with the terms of the agreement.

ViewDS, ViewDS Access Presence and ViewDS Access Sentinel are trademarks of ViewDS Identity Solutions

Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation.

All other product and company names are trademarks or registered trademarks of their respective holders.

Copyright © 1995-2016 ViewDS Identity Solutions

ABN 19 092 422 476

Contents

| | |
|--|---------------|
| Chapter 1 About this guide | 3 |
| Who should read this guide | 3 |
| Conventions | 3 |
| Related documents | 4 |
| How this guide is organized | 4 |
| Chapter 2 System overview | 5 |
| Overview of Access Presence..... | 5 |
| Overview of the Printing DUA..... | 10 |
| Chapter 3 Configuration files..... | 11 |
| Access Presence configuration files | 11 |
| ViewDS configuration file | 15 |
| Additional Access Presence files | 25 |
| Configuring for printing..... | 25 |
| Configuring for global changes | 26 |
| Chapter 4 Access Presence templates | 27 |
| Common tags and arguments | 28 |
| Authentication template..... | 29 |
| Welcome template | 30 |
| Search Forms template | 32 |
| Search Results template | 36 |
| Expanded Entry template | 42 |
| Error template | 51 |
| Modify template | 53 |
| Modify Value Form template | 54 |
| Add template | 56 |
| Print Form template..... | 57 |
| Print template | 58 |
| New Password template..... | 58 |
| Assign Password template | 59 |
| Request Remove Entry template | 60 |
| Global changes templates..... | 60 |
| Target-object cache templates..... | 62 |

| | |
|--|----------------|
| Chapter 5 Format file | 69 |
| Location and syntax | 69 |
| Assessment order and examples | 73 |
| URI and link tags..... | 75 |
| Display name tags..... | 76 |
| Target object tags | 80 |
| Alternative hierarchy tags..... | 81 |
| Approval process tags..... | 82 |
| Chapter 6 Server-side attributes | 84 |
| Important note | 84 |
| Concepts | 84 |
| DUA presentation operational attributes | 86 |
| Preprocessing functions | 109 |
| User operational attributes | 115 |
| Approval process operation attributes..... | 116 |
| New entry operation attributes | 117 |
| Other operational attributes..... | 119 |
| Chapter 7 Printing DUA | 125 |
| Running the Printing DUA | 125 |
| Input script syntax | 126 |
| Supported attribute syntaxes..... | 136 |
| Chapter 8 Printing DUA scripts | 137 |
| Scripts | 137 |
| phonelist.ds | 137 |
| unitlist.ds | 139 |
| executivelist.ds | 141 |
| mailinglist.ds..... | 142 |
| staffdetails.ds | 143 |
| Chapter 9 Advanced features | 147 |
| Configuring proxy authorization for 'single sign on' | 147 |
| Configuring related-entry workflow | 150 |
| Configuring the approval process | 151 |

Chapter 1

About this guide

This guide provides information about two ViewDS user interfaces, *Access Presence* and the *Printing DUA*. Access Presence is a web-based DUA that allows users to search and manage directory data. The Printing DUA produces reports containing directory data, which can be generated from Access Presence.

This chapter has the following sections:

- Who should read this guide
- Conventions
- Related documents
- How this guide is organized

Who should read this guide

Read this guide if you are responsible for developing, modifying or configuring an implementation of Access Presence or the Printing DUA.

You will need to be familiar with HTML, and knowledge of a web-based scripting language is advantageous. You should also be familiar with Access Presence from a user's perspective (the *ViewDS Installation and Configuration Guide* includes an introductory tutorial).

For information about configuring ViewDS for Access Presence, refer to the *ViewDS Installation and Configuration Guide*.

Conventions

The Access Presence tags are in Backus-Naur Form (BNF) and displayed as follows.

```
<VFDeleteHref [confirm] [html=1*CHAR] [id=1*CHAR]>
```

Where:

- | | |
|---------|--|
| <> | Encloses a tag and its arguments. |
| [] | Encloses an optional argument. |
| 1*CHAR | A value comprising one or more characters. |
| 1*DIGIT | A value comprising one or more digits. |
| | Separates the allowable values of an argument. |

For example, consider the following tag:

```
<ExampleTag id=1*DIGIT [confirm] [html=1*CHAR] [select=on|off]>
```

The name of the tag is `ExampleTag` and it has the following arguments:

- `id=1*DIGIT` – this argument must be declared and set to a value comprising one or more digits.
- `[confirm]` – this is an optional argument that does not require a value and therefore acts as a flag.
- `[html=1*CHAR]` – an optional argument, but if declared its value must comprise one or more characters.
- `[select=on|off]` – an optional argument, but if declared its value must be either `on` or `off`.

Related documents

As well as this guide, the ViewDS document set includes the following:

- *ViewDS Installation and Operations Guide*
- *ViewDS Technical Reference Guide: Directory System Agent*
- *Installation and Reference Guide: ViewDS Access Sentinel*
- *Installation and Reference Guide: ViewDS Access Proxy*
- *ViewDS Management Agent Help*

How this guide is organized

Chapter 1: About this guide

Provides a brief overview of this guide.

Chapter 2: System overview

Provides an overview of the main components of Access Presence and the Printing DUA.

Chapter 3: Configuring Access Presence

Describes the Access Presence configuration files and parameters.

Chapter 4: Access Presence templates

Describes the Access Presence templates and their tags.

Chapter 5: Format file

Describes the format file, its syntax and tags.

Chapter 6: Server-side attributes

Describes the server-side attributes that relate to Access Presence.

Chapter 7: Printing DUA

Describes the syntax and use of the Printing DUA scripting language.

Chapter 8: Printing DUA scripts

Describes the Printing DUA scripts that are supplied with ViewDS.

Chapter 9: Advanced configuration

Includes procedures to configure for 'single sign on', related-entry workflow and the approval process.

Chapter 2

System overview

This chapter provides an overview of the Access Presence web DUA and the Printing DUA, their major features and where they fit into the ViewDS architecture. You will need this background information before adapting either.

This chapter has the following sections:

- Overview of Access Presence
- Overview of Printing DUA

Overview of Access Presence

This subsection describes the main components of Access Presence and provides an overview of its functionality.

Main components

Access Presence is the ViewDS web-based client. Figure 1 shows Access Presence, the template and format files it uses to construct web pages, the configuration files it accesses, and the Directory System Agent (DSA) and ViewDS Management Agent.

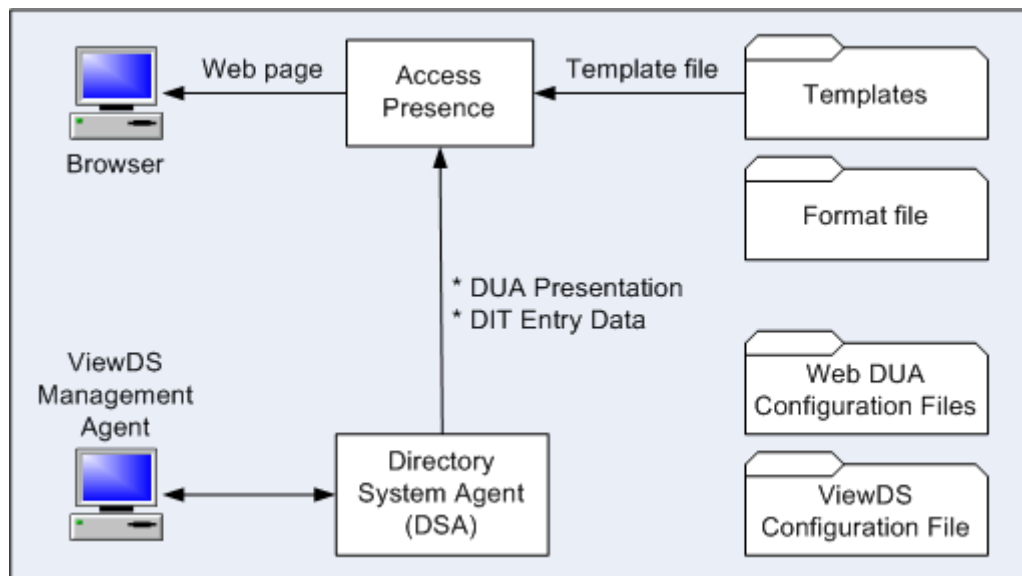


Figure 1: Access Presence components

Access Presence constructs web pages containing directory data obtained from the DSA. How this data is presented is defined by *template files*, the *format file*, and the *DUA presentation* data stored by the DSA. Additional configuration and presentation data is stored in the Access Presence *configuration files* and the *ViewDS configuration file*.

Templates

The Access Presence user interface comprises twelve web pages. Each is constructed from a template file, which is a text file containing standard HTML tags plus proprietary Access Presence tags.

When a user requests a particular page, Access Presence substitutes the proprietary template tags for HTML tags and other information. Some of the Access Presence tags are replaced by directory data, which Access Presence obtains from the DSA, while others are replaced by links to Access Presence functions or pages.

The templates and tags are described in Chapter 4 of this guide.

Format file

The format file contains Access Presence tags that give a finer level of detail than is possible through the tags in the Search Results and Expanded Entry templates.

The format file and its tags are described in Chapter 5 of this guide.

Access Presence configuration files

There are three Access Presence configuration files:

- Extra template file – identifies extra templates that can be used in place of the standard Expanded Entry template.
- MIME mapping file – defines MIME types to associate with attribute types. It allows users to download attributes of any type to a file of the correct type.
- Post-processing command file – defines the processing to perform on an attribute value, and is used to generate URIs from attribute values.

The above files are described in Chapter 3 of this guide.

ViewDS configuration file

Access Presence reads parameters in the ViewDS configuration file. These parameters are described in Chapter 3 of this guide.

Directory System Agent

The DSA provides the following to Access Presence:

- directory data – which Access Presence uses to replace template tags.
- Basic Access Control Items – these access controls can be assigned to users to control their access to objects and attributes in the directory. They can be defined using the Stream DUA or ViewDS Management Agent.
- DUA presentation data – which defines, for example, the appearance of entries, attributes and Search Forms. This data is stored in operation attributes, which can be managed through either the Stream DUA (see Chapter 6) or the ViewDS Management Agent (see below).

ViewDS Management Agent

The ViewDS Management Agent allows you to manage the DUA presentation data stored by the DSA. This data defines, for example, the appearance of attributes, object classes and Search Forms.

For information about managing presentation data through the ViewDS Management Agent, see the topics listed under *Defining DUA presentation* in the application's help system.

Main functionality

The main functionality of Access Presence includes the following:

- Search forms
- Global changes
- Target objects
- Context attributes
- Alternative hierarchy
- Report printing
- Approval process

Search forms

Access Presence allows the user to select from a choice of Search Forms through which to search the directory. Each form has a definition that includes details such as which attributes are available for the user to search on, and how the results are sorted.

The ViewDS Management Agent allows you to define and manage Search Forms – see the ViewDS Management Agent help topic *View or modify a Search Form*.

Global changes

An Access Presence user can now make a change to an attribute's value in multiple entries – referred to as making a global change. To illustrate, if a user changes an attribute's value globally, all instances of the attribute (within the scope of the change) are then set to the new value.

Before a user can apply global changes, the ViewDS DSA must be configured to enable this functionality (see *Configuring for global changes* on page 26).

Target objects

Several Access Presence pages have links relating to the *target-object cache* – for example, these links include 'Set target object' and 'Show target object'.

The 'Set target object' link allows the user to designate an entry to be a *target object* by placing it in the target-object cache. The user can then apply actions to the target object. For example, when they:

- move an entry – the *target object* will become its new superior
- print a report – the *target object* will be the base object for the report
- click 'Show target object' – the *target object* will be displayed

The entry remains in the target-object cache until the user selects 'Set target object' for another entry.

Alternatively, the user can add multiple entries to the target-object cache (for example, by selecting an 'Add to cache' link). The user can then apply actions to all entries in the cache, such as adding them to a group.

This guide describes the templates and tags that allow the user to use target objects.

Context attributes

These are attributes to which a user can assign default values – they do this by entering values and then 'setting the context' in a Search Form. These default values are then assigned to the appropriate *context attributes* whenever the user subsequently performs a search. That is, until the user removes the default values (by 'clearing the context').

Before the 'set context' functionality can be applied to an attribute, it must first be declared a 'context attribute'. You can do this through the ViewDS Management Agent – see the help topic *Manage context attributes*.

Alternative hierarchy

The entries in a directory are arranged in a directory hierarchy. Access Presence presents this arrangement and can also display alternative hierarchies.

As a simple illustration, consider a directory hierarchy where entries are organised according to their department. Figure 2 shows the members of the Test Department at *Widgets and Springs Incorporated*.

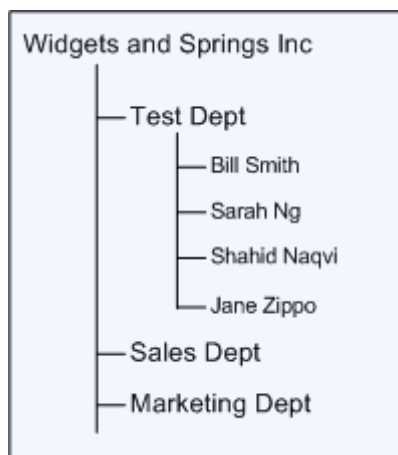


Figure 2: DIT for Widget and Springs Inc

When Access Presence displays the Expanded Entry page for a department member, it also lists the remaining members at the same level in the hierarchy. There is, however, no quick way of seeing who reports to who within the department – Sarah Ng is actually the manager of the Test Department, but this is not conveyed.

The board of Widgets and Springs Inc now decide they wanted to see the managerial relationships within their organisation. This can be achieved through Access Presence by incorporating template tags for an alternative hierarchy. In this case, the tags base the alternative hierarchy on the value of each entry's 'ReportsTo' attribute.

With this implementation, when Access Presence displays details of a department member, it also shows the alternative hierarchy. . So, when Sarah Ng's entry is displayed, all employees who report to her – the rest of the testers – are listed as subordinates.

Report printing

ViewDS includes the Printing DUA which runs print scripts that specify the entries to be extracted from the directory and any subsequent manipulation. Users can run these print scripts from the Print Form page of Access Presence.

Approval process

This mechanism imposes an approval process on changes to the directory.

A user is designated either a 'requestor' or an 'approver'. A requestor can submit a request to modify, delete or move an entry. Later, a user with appropriate access rights, an approver, can either approve or reject the request.

To enable the approval process, see *Configuring the approval process* on page 151.

Access Presence Cookies

Access Presence may make use of three different session cookies:

- `vds-session-id` – The session ID of the managed session (see Directory-based Session Management)
- `VF_Pass` – The credentials of the current user
- `VF_TARGET_O` – Details of the target object

Cookie Security

Both `vds-session-id` and `VF_Pass` are `HttpOnly` cookies. This means that on a supported browser, they will be used only when transmitting HTTP (or HTTPS) requests.

All three session cookies used by Access Presence can be set up as Secure cookies. This means that they can only be used via HTTPS, so they are always encrypted when being transmitting from client to server. To use Secure cookies set the `webcookiesecure` configuration file option to `on`.

Session Management

Two different types of session management can be employed by Access Presence:

- Cookie-based
- Directory-based

Cookie-based Session Management

User credentials and a session expiry are stored in the `VF_Pass` session cookie.

To use cookie-based session management `websessionenabled` must be set to `off` and an integer value for `websessionexpiry` must be provided. This value, if greater than 0, represents the permitted lifetime of the `VF_Pass` cookie in seconds. The default value of 0 indicates that the `VF_Pass` cookie has an unlimited lifetime.

Directory-based Session Management

Sessions are maintained as objects within ViewDS and only the session IDs are stored in the `vds-session-id` session cookie.

To use directory-based session management `websessionenabled` must be set to `on` and the credentials of the user account used for managing session objects in the directory must be provided in `websessionuser` and `websessionpassword`.

This user account must have the necessary permissions to be able to maintain `viewDSSessionObject` operational attribute values in their entry. The `proxyAgent` attribute must also be present.

By default the `websessionexpiry` for sessions managed by the directory is set to 28800s (8 hours). This expiry is calculated from the time the user last authenticated to Access Presence. When the session expires the user is redirected to the authentication form.

The `websessioncleanup` configuration file option provides a clean-up mechanism for directory-based sessions. It is an integer with a default value of 86400s (24 hours) and represents the maximum age of a session object in the directory in seconds. When Access Presence adds or removes a session object for the current user, it checks for any other session objects older than this and removes them as part of the same modify operation.

Overview of the Printing DUA

The input to the Printing DUA is a script and its output is an ASCII file. The script specifies the entries to be extracted and any subsequent data manipulation. The Printing DUA can sort the extracted data and insert data (text, tabs, tags, line breaks, etc.) as required.

The Printing DUA scripts can be invoked from Access Presence or from some other mechanism external to the directory – such as, the command line or a batch file or scheduler.

Input script

The input script to the Printing DUA specifies:

- the data to be extracted
- the order in which the data is to be outputted
- the format of the outputted data

A major feature of the Printing PDU is *flexible output* - this feature enables a user to specify the form of the output information. Therefore, if it is intended that the output should be

- read as plain text, it is generated so that it is as *readable* as possible.
- the input into another computer system or application (such as a word processing application), it is generated in the form required by the target system or application.

Output file

Although the functionality of the Printing DUA is powerful, there may be occasions where it cannot quite produce the precise output required. In this case, it will be necessary to output data close to the requirement and then process it further with any general purpose programming language.

When a formatted document is required, the Printing DUA's output can be fed into a target application. To accommodate this it is usually necessary for the Printing DUA to generate 'tagged data'. The target application can then be used to publish the document – for example, by setting the fonts, headers, footers, table of contents, and adding text and graphics as required.

Chapter 3

Configuration files

This chapter describes the configuration files accessed by Access Presence, and also describes how to configure for printing.

It has the following sections:

- Access Presence configuration files
- ViewDS configuration file
- Additional Access Presence files
- Configuring for printing
- Configuring for global changes

Access Presence configuration files

This section describes the following configuration files that are specific to Access Presence:

- Extra template file
- MIME mapping file
- Post-processing command file

Extra template file

The *extra template file* identifies 'extra templates' that can be used in place of the standard Expanded Entry template (see page 42) or Search Results template (see page 36).

Many of the Access Presence tags generate a URI – for example, `VFAuthURI` and `VFSearchURI`. These tags have an optional `id` argument that identifies an extra template to be included in the URI in place of the standard Expanded Entry template. Whenever a user subsequently views the Expanded Entry page, it is generated using the extra template identified in the URI. To revert to the standard Expanded Entry template, the `id` argument must be set to ``_default``.

Each extra template can also be associated with a *template-specific format file*, which is defined in the same way as the standard format file (see page 69). The extra template then uses its template-specific format file plus the standard format file. (The definitions in the template-specific format file, however, take precedence over those in the standard format file.)

NOTE: The page generated from an extra template can also be accessed from a standard Access Presence page using the `VFURI` or `VHref` tags (see page 75).

Location

The file is identified by the configuration-file parameter `webExtraTemplateFile` (see page 24).

Syntax

Each line in the extra template file identifies an extra template:

```
identifier%content_type%disposition% extraTemplate extraFormat
```

Where the % character is a delimiter, and:

- `identifier` is an alphanumeric string without any spaces.
- `content_type` is optional and identifies the content type (for example, `%text/xml`) to appear in the HTTP content-type header of the page generated from the template. If no content type is declared, the default `%text/html` applies. If the content type is declared starting with `%text/`, then `;charset=...` is appended to the content-type header and the character encoding is set to UTF-8.
- `disposition` is optional and identifies the HTTP content-disposition header of the page generated from the template. It can include spaces and quotation marks, but cannot include a % character as this is used as a delimiter.
- `extraTemplate` is the path and name of a template file relative to the webdir directory (by default, `${VFHOME}/webdir` or `%VFHOME%\webdir`).
- `extraFormat` is optional and identifies the path and name of a template-specific format file relative to the webdir directory (by default, `${VFHOME}/webdir` or `%VFHOME%\webdir`).

A path and name that contains space characters must be enclosed by double-quote or apostrophe characters. These characters can be escaped using the backslash character.

The extra template file contains a maximum of 25 identifiers. (This is because Access Presence maps each identifier to a character between 'b' and 'z', which it then includes in the URI for the subsequent Expanded Entry page.)

Examples

Consider the following entries in the extra template file:

```
treeExpanded  /tmpl/altTmp/tree_expanded_entry.html
maxExpanded   "/tmpl/altTmp/max_detail.html"
minExpanded   /tmpl/altTmp/min_detail.html      min_formatFile
xmlHierarchy%text/xml  /tmp/altTmp/xmler.xml
```

They declare three extra Expanded Entry templates identified by the strings `treeExpanded`, `maxExpanded` and `xmlHierarchy`.

Now, consider the following Access Presence tag:

```
<VFWelcomeURI [id=*CHAR]>
```

This tag can be used in a template to generate a URI to the Welcome page:

```
<A HREF="<VFWelcomeURI">">Welcome page</a>
```

It can also be used to identify an alternative Expanded Entry template to be included in the URI:

```
<A HREF="<VFWelcomeURI id=treeExpanded">">My details</a>
```

Access Presence will then use the template file `tree_Expanded_Entry.html` when the user views the Expanded Entry page.

NOTE: To revert to the standard Expanded Entry page, a template must set the `id` argument to `'_default'`.

The last example above, `xmlHierarchy`, is slightly different to the other two examples. It defines that the MIME type to be generated from the template is XML.

MIME mapping file

The MIME mapping file defines MIME types to associate with attribute types. It allows users to download attributes of any type to a file of the correct type.

Location

The location is set by the configuration-file parameter `webAttMIMEMappingFile` (page 21).

Syntax

The file comprises one or more lines that take the following format:

```
entry ::= attributeReference wp mimeType [ wp message ]
        eol [ tab disposition eol ]
wp ::= ' ' | '\t'
eol ::= [ '\r' ] '\n'
attributeReference ::= attributeName [ "."
componentReference ]
```

Where:

- `attributeName` – the name of an attribute type.
- `mimeType` – the MIME type used as the content-type HTTP header when the attribute value is downloaded to a browser.
- `componentReference` – the component reference of a field within the attribute value. The component reference syntax is defined in the following RCF: Legg, S (2004), *LDAP & X.500 Component Matching Rules*, <http://tools.ietf.org/html/rfc3687>. A component reference allows any field within a syntax defined as an ASN.1 type to be referenced.
- `disposition` – a string used as the content-disposition HTTP header when the attribute value is downloaded to a browser.
- `message` – the message displayed in the browser to represent this attribute value.

The entries are evaluated in the order they occur in the file.

The `attributeReference` is a Boolean assertion on the attribute value. The first entry with an assertion in the `attributeReference` that evaluates to true provides:

- an attribute-type name, which will match any value of the identified attribute;
- an attribute-type name with a component reference of a field in the value of the attribute, which will match any value of the attribute in which the referenced field is present;
- an attribute-type name with a component reference of a bit in a BitString field in the attribute value, which will match any value of the attribute in which the referenced BitString is present and the identified bit is set.

The `message` field may contain some special character sequences, which Access Presence will replace with certain values. These sequences are:

- `%s` – which will be replaced with the URI required to download the value. (Previous versions of this file used a single `%` character for this purpose.)
- `%{...}` – where the braces contain a comma-separated list of component references. This expression will be replaced by the string value of a field in the attribute value identified by the component reference list. The component references are evaluated from left to right until a reference to a field present in the current value is found. Where the identified field is a string field, the string is converted into an appropriate code page (currently ISO-8859-1); otherwise the field is encoded into a string (of the appropriate code page) using the generic string encoding rules described in the following RCF: Legg, S (2003), *Generic String Encoding Rules for ASN.1 Types*, <http://tools.ietf.org/html/rfc3641>.
- `%%` – which will be replaced with an explicit `%` character.

The `disposition` field may also contain some special character sequences that Access Presence will replace with certain values. The `%{...}` and `%%` character sequences described for the message field are supported for the `disposition` field.

Examples

These three examples are from the MIME mapping file, `att.mime`, supplied with the demonstration directory, `Deltawing`.

1. This example tells Access Presence that attributes of the type `photo` should be treated as the MIME type `image/jpeg`:

```
photo image/jpeg
```
2. This example tells Access Presence to display attributes of type `document` as the MIME type `application/pdf`.

```
document application/pdf
```
3. This example maps the attribute type `cACertificate`:

```
cACertificate application/pkix-cert Download certificate for <A
HREF="%s">{%toBeSigned.subject.rdnSequence.-1.*.value}</A>
application; filename="%{toBeSigned.subject.rdnSequence.-
1.*.value}"
```

Where:

- `cACertificate`
The attribute type in the directory.
- `application/pkix-cert`
The MIME type.
- `Download certificate for {%toBeSigned.subject.rdnSequence.-1.*.value}`
The component reference displayed by the browser.
- `application; filename="%{toBeSigned.subject.rdnSequence.-1.*.value}"`
The disposition which defines the file name to which the certificate will be saved when downloaded by the user.

Post-processing command file

This file defines the processing to perform on an attribute value, and is used to generate URIs from attribute values.

Location

The file's location is set by the configuration-file parameter `webPostProcessCmdFile` (see page 21).

Syntax

The file comprises one or more lines as follows:

```
attribute-name [output-string] [attribute-names]
```

Where:

- `attribute-name` – is the name of an attribute type.
- `output-string` – is the string to display in place of the attribute value.
- `attribute-names` – is a list of attributes whose values are substituted for '%' or '\$' characters in `output-string`. A value substituted for a '%' has characters that are illegal in URIs suitably escaped, so use a '%' if the value is to appear in a URL. A value substituted for a '\$' is not altered.

For example (a single line):

```
telephoneNumber
"<A HREF=http://www.deltawing.com.au/call.acgi?BNUM=%&STYLE
=ST1&BNAME=%&BADDR =%&LANG=e>$</A>" telephoneNumber commonName
location telephoneNumber
```

ViewDS configuration file

This section describes the parameters in the ViewDS configuration file that relate to Access Presence. By default, the configuration file is `${VFHOME}/setup/config` or `%VFHOME%\setup\config`.

All paths specified by the parameters are either absolute or relative to the `webdir` directory.

The parameters are grouped as follows:

- General parameters
- Authentication parameters
- Search Form parameters
- Search Results parameters
- Search parameters
- Display and format parameters
- Image parameters
- Template location parameters
- Other template-related parameters

General parameters

| | |
|--------------------------------------|--|
| <code>dsaAddress</code> | <p>The default base address and protocol for connections to the DSA.</p> <p>If Access Presence is installed on a different host to the one running the DSA, the <code>dsaAddress</code> should be set to the TCP/IP address and port of the DSA. However, if this is not possible, the parameter <code>webduaaddress</code> can be set to an address and protocol for Access Presence to use.</p> |
| <code>baseEntry</code> | <p>The Distinguished Name (DN) of the subschema administrative point in Stream DUA notation (see <i>Technical Reference Guide: Directory System Agent</i>).</p> <p>For example: <code>baseentry = { 0 "Deltawing" }</code></p> <p>This parameter allows Access Presence to retrieve schema and DUA Presentation data. Access Presence also uses this parameter as the default base entry for searches.</p> <p>No default.</p> |
| <code>webdir</code> | <p>The directory containing the Access Presence configuration and data files.</p> <p>Default: <code>webdir = \${VFHOME}/webdir</code></p> |
| <code>webduaAddress</code> | <p>The base address and protocol for connections to the DSA by Access Presence. The parameter enables support for OSI protocol connections over RFC1006, and can be set to a lightweight stack address, IDM protocol address or OSI protocol address.</p> <p>If this parameter has been set, then it defines the protocol and address that Access Presence uses to connect to the DSA. Otherwise, Access Presence connects to the DSA using the default protocol and address defined by the <code>dsaAddress</code>.</p> |
| <code>webDUAParamPath</code> | <p>The path of the directory where Access Presence stores cached DUA Presentation information. This path must be unique for each instance of Access Presence.</p> <p>Default:</p> <p><code>webDUAParamPath = \${VFHOME}/webdir/conf</code></p> |
| <code>webMaxRequestsPerServer</code> | <p>The maximum number of requests that an instance of Access Presence will handle before it terminates and is restarted.</p> <p>Default: <code>webMaxRequestsPerServer = 50</code></p> |
| <code>webCGITimeLimit</code> | <p>The maximum number of seconds that an instance of Access Presence will handle one request before it terminates and is restarted.</p> <p>Default: <code>webCGITimeLimit = 120</code></p> |

| | |
|--------------------------------|---|
| <code>webUseUuid</code> | <p>Determines how Access Presence references entries in the URLs it generates:</p> <ul style="list-style-type: none"> • <code>on</code> – use the <code>entryUUID</code> attribute (see <i>RFC 4530</i>) to generate a URI to identify an entry. When the <code>entryUUID</code> attribute is unavailable, the entry's DN is used. • <code>off</code> – use the DN to generate a URI to identify an entry. This option is available for compatibility with ViewDS prior to Version 7.0. <p>Default: <code>webUseUuid = on</code></p> |
| <code>webSchemaArea</code> | <p>The Distinguished Name of a schema administrative point from which Access Presence obtains schema and parameterisation information. This parameter is of use for a distributed environment in which schema and parameterisation information is unavailable in some areas of the DIT.</p> <p>There is no default for this parameter.</p> |
| <code>webcookiesecure</code> | <p>A Boolean that defines whether the Secure property is set for the <code>VF_Pass</code>, <code>VF_TARGET_O</code> and <code>vds-session-id</code> cookies.</p> <p>Default: <code>webcookiesecure = off</code></p> |
| <code>websessionexpiry</code> | <p>If <code>websessionenabled = off</code>, then this is an integer value that, if greater than 0, represents the permitted lifetime of the <code>VF_Pass</code> cookie in seconds. The default value of 0 indicates that the cookie has an unlimited lifetime.</p> <p>If <code>websessionenabled = on</code>, then this is an integer value that represents the permitted lifetime of a log in session in seconds. The default value of 28800 indicates that the session has an 8 hour lifetime.</p> |
| <code>websessionenabled</code> | <p>A Boolean that controls whether directory-based session management is used. When this is set to <code>on</code> Access Presence will use session objects stored in the directory. The <code>VF_Pass</code> cookie will be deleted and a new cookie called <code>vds-session-id</code> will be used that contains only a randomly generated session identifier. This cookie will have the <code>HttpOnly</code> attribute set and will have the <code>Secure</code> attribute set if the <code>webcookiesecure</code> configuration file option is set to <code>on</code>.</p> <p>Default: <code>websessionenabled = off</code></p> |
| <code>websessionuser</code> | <p>A string containing the LDAP string encoded DN or the username of the account Access Presence should use for managing session objects in the directory.</p> <p>This configuration file option is only required if <code>websessionenabled = on</code></p> |

| | |
|---------------------------------|--|
| <code>websessionpassword</code> | <p>A string containing the password of the account identified in <code>websessionuser</code>.</p> <p>This configuration file option is only required if <code>websessionenabled = on</code></p> |
| <code>websessioncleanup</code> | <p>An integer that represents the maximum age in seconds of session objects in the directory. This is used to clean-up sessions managed by the directory. When Access Presence adds or removes a session object for the current user, it checks for any other session objects older than this period and removes them.</p> <p>Default: <code>websessioncleanup = 86400 (24 hours)</code></p> |

Authentication parameters

| | |
|------------------------------------|--|
| <code>webRequireLogin</code> | <p>A Boolean that defines whether Access Presence performs login authentication each time the directory is accessed.</p> <p>Default: <code>webRequireLogin = on</code></p> |
| <code>webBindUser</code> | <p>Defines the user name and password that Access Presence uses to bind to the directory. It is used if <code>webRequireLogin</code> is off or <code>webProxyUser</code> is on.</p> <p>Example: <code>webBindUser = vfsuper passwd</code></p> |
| <code>webProxyUser</code> | <p>A Boolean that enables proxy authentication.</p> <p>Default: <code>webProxyUser = off</code></p> |
| <code>webProxyAuthAttribute</code> | <p>Access Presence uses this parameter to map an identity determined by a web server to an entry in the ViewDS directory. It is used when <code>webProxyUser</code> is on.</p> <p>Default: <code>webProxyAuthAttribute = userName</code></p> |
| <code>webAnonymous</code> | <p>Governs whether Access Presence requires the user to provide authentication information to log in to the directory. When this parameter is set to <code>on</code>, Access Presence will not initially prompt for a user login. If it does prompt for authentication information (see the <code>VFAuthURI</code> tag on page 30), it will permit the Access button to be selected without any information being provided in the other input fields. The default setting for this parameter is <code>off</code>.</p> <p>This parameter differs from <code>webRequireLogin</code> by explicitly supporting anonymous logins, whether the authentication form is used or not. It is recommended that <code>webRequireLogin</code> is set to <code>on</code> when this parameter is used.</p> <p>Default: <code>webAnonymous = off</code></p> |
| <code>webRemoteUser</code> | <p>Sets the name of the environment variable used by a web server to pass an authenticated identity to a CGI during proxy authorization (see page 147).</p> <p>Default: <code>webRemoteUser = REMOTE_USER</code></p> |

`webUsePasswordPolicy` A Boolean that defines whether Access Presence should use password policy controls in requests.
 Default: `webUsePasswordPolicy = off`

Search Form parameters

`webSelectableField` Determines whether the Search Form page present search fields that are:

- restricted to one specific attribute (`off`)
- allow an attribute to be selected from a drop-down list (`on`)

Additionally, if a set of possible values is declared for an attribute in its schema, and this parameter is `off`, the Search Form presents the values in a drop-down list. (See *Drop-down lists* on page 32.)

Default: `webSelectableField = on`

`webNumDefaultFields` Defines the number of active search fields which Access Presence will show on the Search Form page. An active field is a field which has an attribute assigned to it. This parameter is ignored if `webSelectableField` is `off`. Its maximum value is 10.

Default: `webNumDefaultFields = 5`

`webNumDisplayFields` Defines the total number of search fields to display. If `webSelectableField` is `on`, Access Presence will show a number of inactive fields equal to `webNumDisplayFields` minus `webNumDefaultFields`. Each inactive field is labelled '-----' in the search form. These are the extra fields which the user can use if `webNumDefaultFields` number of search fields are not enough for specifying his or her query. Its maximum value is 10.

Default: `webNumDisplayFields = 8`

`webUserOptions` Determines whether Access Presence appends the search options to the Search Form.

Default: `webUserOptions = on`

`webForceLocalSchema` Determines whether Access Presence uses the schema in the default base entry if there is no schema under the base entry associated the current Search Form.

Default: `webForceLocalSchema = off`

`webRelatedSearch Template` Specifies a template to be used instead of the Search Form template during the related-entries workflow (see page 150). When this parameter is unspecified, the template specified by `webSearchTemplate` is used.

Search Results parameters

| | |
|---|--|
| <code>webRelatedSearchResultTemplate</code> | Specifies a template used instead of the Search Results template during the related-entries workflow (see page 150). When this parameter is unspecified, the template specified by <code>webSearchResultTemplate</code> is used. |
| <code>webSortStrategy</code> | <p>Specifies the algorithm used to sort the results displayed on the Search Results page (see page 36). The arguments are:</p> <ul style="list-style-type: none"> • <code>sortKeys</code> – sort the fields by the sort keys in the Search Form. The sort key is defined through the operational attribute <code>searchOptions</code> (see page 101) – for example, through the ViewDS Management Agent (see the help topic <i>View or modify a Search Form</i>). • <code>searchFields</code> – sort the fields using the first completed field on the Search Form as the initial key, then use the sort keys in the Search Form. • <code>firstValueExactMatch</code> – start the results with any entry where the first value of the search attribute exactly matches the value entered in the Search Form. Then, sort using the <code>searchFields</code> algorithm described above. • <code>anyValueExactMatch</code> – start the results with any entry where any value of the search attribute exactly matches the value entered in the Search Form. Then, sort using the <code>searchFields</code> algorithm described above. <p>Default:</p> <p><code>webSortStrategy = firstValueExactMatch</code></p> |

Search parameters

These parameters define the search options when `webUserOptions` is off, and the default values when `webUserOptions` is on.

| | |
|------------------------------|--|
| <code>webDefSizeLimit</code> | <p>The default size limits. The size limit setting here is only used if there is no user or system-wide setting stored in the directory.</p> <p>Default: <code>webDefSizeLimit = 100</code></p> |
| <code>webDefTimeLimit</code> | <p>Specifies the default time limits. The time limit setting here is only used if there is no user or system-wide setting stored in the directory.</p> <p>Default: <code>webDefTimeLimit = 10</code></p> |
| <code>webShowDetail</code> | <p>Determines whether Access Presence allows the ‘expand’ operation for the entries returned from a successful search. It overrides <code>webDetailOnSingleMatch</code>.</p> <p>Default: <code>webShowDetail = on</code></p> |

| | |
|---|---|
| <pre>webShow = {Sub SubAndLeaves Icon IconAndLabel All}</pre> | <p>This parameter defines what is displayed by the Expanded Entry page.</p> <p>The arguments are:</p> <ul style="list-style-type: none"> • Sub: display all non-leaf entries one level down from the expanded entry. • SubAndLeaves: display all entries one level down from the expanded entry. • Icon: use descriptive icons only as labels to the entry's attributes. • IconAndLabel: use both icon and text description as labels to the entry's attributes. • All: all of the above options apply. <p>All the above options except All may be used together on the same line. By default, none of the options are set.</p> <p>Example: <code>webShow = All</code></p> |
| <pre>webDetailOnSingle Match</pre> | <p>Determines whether Access Presence automatically expands an entry if it is the only match returned by a search.</p> <p>Default: <code>webDetailOnSingleMatch = on</code></p> |
| <pre>webDefExactMatch</pre> | <p>Determines whether Access Presence does normal approximate matches the first time (<code>off</code>); or whether it does an exact match search, and if there are no matches, follows this with an approximate match search. This two-stage search gives a faster response with large databases.</p> <p>Default: <code>webDefExactMatch = off</code></p> |

Display and format parameters

| | |
|---------------------------------------|---|
| <pre>webAttMIMEMappingFile</pre> | <p>The path (relative to the <code>webdir</code> directory) to the MIME mapping file (see page 13).</p> |
| <pre>webPostProcessCmdFile</pre> | <p>The path (relative to the <code>webdir</code> directory) to the Post-processing command file (see page 15).</p> |
| <pre>webTargetObject = on off</pre> | <p>Controls whether the target-object links (see <i>Target-object</i> on page 62) are displayed to all users (<code>on</code>) or just those with either of the following (<code>off</code>):</p> <ul style="list-style-type: none"> • the super-user privilege of ViewDS Access Control (see the ViewDS Management Agent help topic <i>Set the ViewDS Access Control for an entry</i>) • the Printing capability of the default entitlements (see the ViewDS Management Agent help topic <i>View or modify default entitlements for DUA users</i>) |

Image parameters

All images should be .gif files for general compatibility.

| | |
|-------------------------------|---|
| <code>webBackGroundPic</code> | Path and file name for the background image displayed by the Access Presence pages. |
| <code>webHeaderPic</code> | Path and file name for the header image displayed on all Access Presence pages. |
| <code>webHeaderPicDesc</code> | Text description of the header image (for text-only browsers). |
| <code>webFooterPic</code> | Path and file name for the footer image displayed on all Access Presence pages. |
| <code>webFooterPicDesc</code> | Text description of the footer image (for text-only browsers). |

Template location parameters

The default Access Presence pages are built-in. However, their equivalent HTML source is provided under the 'suggested' file names below.

| | |
|--------------------------------------|---|
| <code>webAuthTemplate</code> | Path and file name of the Authentication template (see page 28). Suggested: <code>webAuthTemplate = tpl/auth.html</code> |
| <code>webWelcomeTemplate</code> | Path and file name of the Welcome template (see page 30). Suggested: <code>webWelcomeTemplate = tpl/wel.html</code> |
| <code>webSearchTemplate</code> | Path and file name of the Search Forms template (see page 32). Suggested: <code>webSearchTemplate = tpl/s.html</code> |
| <code>WebSearchResultTemplate</code> | Path and file name of the Search Results template (see page 36). Suggested: <code>webSearchResultTemplate = tpl/sr.html</code> |
| <code>WebExpandResultTemplate</code> | Path and file name of the Expanded Entry template (see page 42). Suggested: <code>webExpandResultTemplate = tpl/er.html</code> |
| <code>webErrorTemplate</code> | Path and file name of the Error template (see page 47). Suggested: <code>webErrorTemplate = tpl/error.html</code> |
| <code>webModifyTemplate</code> | Path and file name of the Modify template (see page 53). Suggested: <code>webModifyTemplate = tpl/mod.html</code> |

| | |
|------------------------------------|--|
| webModifyValueTemplate | <p>Path and file name of the Modify Value template (see page 54). Suggested:</p> <pre>webModifyValueTemplate = tpl/modval.html</pre> |
| webAddTemplate | <p>(Path and file name of the Add template (see page 54). Suggested:</p> <pre>webAddTemplate = tpl/add.html</pre> |
| webPrintFormTemplate | <p>Path and file name of the Print Form template (see page 57). Suggested: webPrintFormTemplate =</p> <pre>tpl/pf.html</pre> |
| webPrintTemplate | <p>Path and file name of the Print template (see page 58). Suggested: webPrintTemplate =</p> <pre>tpl/pr.html</pre> |
| webNewPasswordTemplate | <p>Path and file name of the New Password template (see page 58). Suggested: webNewPasswordTemplate =</p> <pre>tpl/npw.html</pre> |
| webAssignPassword Template | <p>Path and file name of the Assign Password template (see page 59). Suggested: webAssignPasswordTemplate =</p> <pre>tpl/apw.html</pre> |
| webGlobalChangeRequest Template | <p>Path and file name of the Global Change Request template (see page 61). Suggested: webGlobalChangeRequestTemplate =</p> <pre>tpl/gcrequest.html</pre> |
| webGlobalChangeConfirm Template | <p>Path and file name of the Global Change Confirm (see page 61). Suggested: webGlobalChangeConfirmTemplate =</p> <pre>tpl/gcconfirm.html</pre> |
| webGlobalChangeResult Template | <p>Path and file name of the Global Change Results (see page 62). Suggested: webGlobalChangeResultTemplate =</p> <pre>tpl/gcresult.html</pre> |
| WebShowTargetObjects Template | <p>Path and file name of the <i>Show Target Objects template</i> (see page 63). Suggested: webShowTargetObjectsTemplate =</p> <pre>tpl/showtarget.html</pre> |
| webSelectTargetObject Template | <p>Path and file name of the <i>Select Target Object template</i> (see page 64). Suggested: webSelectTargetObjectTemplate =</p> <pre>tpl/selecttarget.html</pre> |

| | |
|--|--|
| <code>webSelectImportsTemplate</code> | <p>Path and file name of the <i>Select Entries to Import template</i> (see page 65).</p> <p>Suggested: <code>webSelectImportsTemplate = tmpl/selectimports.html</code></p> |
| <code>webSelectRemovalsTemplate</code> | <p>Path and file name of the <i>Select Entries to Remove template</i> (see page 66).</p> <p>Suggested: <code>webSelectRemovalsTemplate = tmpl/selectremovals.html</code></p> |
| <code>WebAltAddSelectionTemplate</code> | <p>Path and file name of the <i>Select Entries to Add to Alternative Hierarchy template</i> (see page 66).</p> <p>Suggested: <code>WebAltAddSelectionTemplate = tmpl/altaddselection.html</code></p> |
| <code>webAltMoveSelectionTemplate</code> | <p>Path and file name of the <i>Select Entries to Move in Alternative Hierarchy template</i> (see page 67).</p> <p>Suggested: <code>webAltMoveSelectionTemplate = tmpl/altmoveselection.html</code></p> |
| <code>webRequestRemoveentryTemplate</code> | <p>Path and file name of the <i>Request Remove Entry template</i> (see page 60).</p> <p>Suggested: <code>webRequestRemoveEntryTemplate = tmpl/requestremoveentry.html</code></p> |

Other template-related parameters

| | |
|---|--|
| <code>defaultCharSet = character_set</code> | Deprecated. |
| <code>webTemplAttribute = attribute_name</code> | <p>This parameter allows an alternative Expanded Entry template to be invoked for a specific directory entry. The entry must include an attribute whose name matches the value of the <code>webTemplAttribute</code> parameter. The value of this attribute should be the name of the alternative Expanded Entry template to be invoked.</p> <p>For example, consider a directory entry that includes the attribute <code>specialTemplate</code> whose value is the name of an Expanded Entry template for people in the Finance Department:</p> <pre>specialTemplate = tmpl/financeExpEntry</pre> <p>To invoke the template, the <code>webTemplAttribute</code> parameter must be set to the name of the above attribute:</p> <pre>webTemplAttribute = specialTemplate</pre> <p>Different entries can include the <code>specialTemplate</code> attribute with values that invoke different templates.</p> |
| <code>webExtraTemplateFile</code> | The path (relative to the <code>webdir</code> directory) to the Extra template file (see page 11). |

| | |
|----------------------------|--|
| | For Deltawing, the default is: <code>tmpl/extra.lst</code> |
| <code>webFormatFile</code> | <p>The path (relative to the <code>webdir</code> directory) to the format file (see page 69).</p> <p>A supplied example is: <code>tmpl/format.lst</code></p> |
| <code>webstartpage</code> | <p>Declares which page is displayed after a user has been authenticated.</p> <p>By default, a user authenticates by entering their user name and password on the Authentication page, and Access Presence then displays the Welcome page.</p> <p>This default behaviour can be changed by setting the <code>webstartpage</code> parameter to one of the following arguments:</p> <ul style="list-style-type: none"> • <code>search</code> – the Search Form is displayed after authentication • <code>baseentry</code> – the Expanded Entry page for the base entry is displayed after authentication • <code>userentry</code> – the Expanded Entry page for the authenticated user is displayed after authentication (if there is no entry for the user, the base entry is displayed) <p>If this parameter is not declared (or set to <code>welcome</code>) the default behaviour applies.</p> |

Additional Access Presence files

The demonstration directory, Deltawing, uses additional files in the following directories:

- `webdir/help` – this directory contains help files for Access Presence.
- `webdir/icons/newicons` – this directory contains icons that are included in Expanded Entry pages.
- `webdir/tmpl` – this directory contains the sample templates described in Chapter 4.

Configuring for printing

A user can invoke a Printing DUA input script from a menu on the Print Form page (see page 57) and to select a base object for the script.

The content of the menu is configured in the following file:

`${VFHOME}/setup/printconfig`

Each line of this file contains the following fields in this order, separated by the `\` character:

- label - a descriptive label for the report
- script name - the name of the PDUA script used to extract the data from the report (in `${VFHOME}/print`).

- post processing script - the name of an optional Shell script to process the extracted data (in `${VFHOME}/print`). This field may be empty.
- content type - the Content-Type value to use when delivering the report to the browser. This field may be empty.
- content disposition - the Content-Disposition value to use when delivering the report to the browser. This field may be empty.

The supplied `printconfig` file lists the supplied scripts (see Chapter 8).

Configuring for global changes

Before an Access Presence user can apply global changes to the values of attributes, you must configure the ViewDS DSA to enable this functionality. You can configure the DSA using either the Stream DUA or ViewDS Management Agent.

For the ViewDS Management Agent, see the help topic *Configure for global changes*.

For the Stream DUA:

1. Enable an attribute for global changes by setting its `globallyChangeable` flag in the operational attribute `attributePresentation` (see page 87).
2. Allow users to apply global changes by setting the `globalChanges` flag in either `defaultEntitlement` (see page 106) or `userEntitlement` (see page 115).

Chapter 4

Access Presence templates

This chapter describes the Access Presence templates and tags. It includes example code from the demonstration directory supplied with ViewDS, Deltawing. For information about using Deltawing, see the *ViewDS Installation and Operations Guide*.

This chapter has the following sections:

- Common tags and arguments
- Authentication template
- Welcome template
- Search Forms template
- Search Results template
- Expanded Entry template
- Error template
- Modify template
- Modify Value Form template
- Print Form template
- Print template
- New Password template
- Assign Password template
- Request Remove Entry template
- Global changes templates
- Target-object cache templates

The file names and locations of the Access Presence templates are set in the ViewDS configuration file (see *Template location* parameters on page 22).

Common tags and arguments

Common template tags

These tags can be used in any Access Presence template, except the Authentication template.

<VFUserID>

This tag is replaced by the display name of the user currently logged in.

<VFPASSWORDPolicyExpiry [before=1*CHAR] [after=1*CHAR]>

This tag is replaced by the number of seconds until the current user's password is due to expire and the account is locked. It generates values only when the password policy indicates that a password is due to expire. The tag takes two optional arguments *before* and *after* which can be used to provide HTML that will be put before and after the tag value.

<VFPASSWORDPolicyGraceLogins [before=1*CHAR] [after=1*CHAR] >

This tag is replaced by the number of grace logins permitted for the current user before their account is locked. Grace logins are provided after a password expires to permit the user to change their password. The tag generates values only when a password has expired and the grace logins apply. It takes two optional arguments *before* and *after* which can be used to provide HTML that will be put before and after the tag value.

<VFUserURI [id=*CHAR]>

This tag is replaced by the URI for the Expanded Entry page displaying the entry for the currently authorized user.

The *id* argument is common to many tags and is described on page 29.

<VFDN>

This tag is replaced by the LDAP DN of the entry currently displayed by the Expanded Entry page.

<VFSetSearchForm name=1*CHAR>

This block tag allows the Search Form to be specified in a template. Within the scope of the `VFSetSearchForm` tag, the Search Form selected on the Welcome page is overridden by the Search Form specified in the tag's *name* argument.

The block must be terminated with an end tag:

```
<VFSetSearchForm name="Name Search">
  <form action="<VFSearchURI">">
    <!-- HTML for simple search form -->
  </form>
</VFSetSearchForm>
```

<VFBaseDNURI [id=*CHAR]>

This tag is replaced by the URI required to access the Expanded Entry page for the base object.

Common arguments

The following arguments are common to many template tags.

id

The `id` argument identifies an alternative Expanded Entry template to be included in the URI generated by the tag it appears in. The `id` for the alternative template must be declared in the extra template file (see *Extra template file* on page 11).

To revert to having the standard Expanded Entry template in the URI, set the `id` to `'_default'`.

html

The `html` argument should be enclosed in backquotes and should include all the text to appear between the opening `<A HREF` and the closing ``.

To illustrate, the following example would display the text 'Set Password' linked to the Assign Password page:

```
<VFAssPassFormHref html=">Set Password">
```

The default value of the `html` argument is `>`.

confirm

The `confirm` argument invokes a confirmation dialog when the link generated by its tag is selected by a user.

format

This argument identifies a set of format directives in the format file (see page 69). The directives are applied to the information generated by the tag that includes the `format` argument.

scope

This argument changes the scope of a tag. It can be set to:

- `base` – the scope is the base entry and its subordinates.
- `user` – the scope is the entry for the user viewing the page and its subordinates.

Alternatively, it can be set to a specific entry identified by either:

- the value of the entry's `entryUUID` attribute; or
- the LDAP string representation of the entry's Distinguished Name.

Authentication template

This template generates a web page that allows a user to enter either their Distinguished Name or user name plus their password.

The template includes a form containing four Access Presence tags, which are described below.

```
<form method=post action="<VFAuthURI>">
  Username: <input name="<VFAuthUserNameID>"> <br>
```

```

    Password: <input type=password NAME="<VFAuthUserPassID>"> <br>
    <VFAuthReferer>
    <input type=submit value="Access">
  </form>

```

<VFAuthURI [id= *CHAR]>

This tag is replaced by a Uniform Resource Identifier (URI) for the Authentication page. The `id` argument is common to many tags and is described on page 29.

<VFAuthUserNameID>

This tag gives the value for the HTML `NAME` element in the user-name `input` tag.

<VFAuthUserPassID>

This tag gives the value for the HTML `NAME` element in the password `input` tag.

<VFAuthReferer>

This tag ensures that Access Presence returns to the page the user was attempting to view before authentication.

The tag is replaced by an HTML `input` element:

```
<input type='hidden' name='referrer' value=URL>
```

The `value` is equal to the `URL` for the page the user was attempting to view.

Welcome template

This template generates the Welcome page. By default, this is the first page displayed after a user has been authenticated.

NOTE: The configuration-file parameter `webstartpage` (see page 25) can be used to declare another page as the first to be displayed after a user has been authenticated.

The Welcome page displays a list of Search Forms from which the user can select.

The Search Forms are declared through either:

- ViewDS Management Agent – see the help topic *View or modify a Search Form*
- Stream DUA – see the operational attribute *searchOptions* on page 101

The Access Presence tags in the template are included in the following code, and each is described below.

```

<VFBanner> <br>
Version: <VFVersion>
<form method="post" action="<VFSearchFormURI>">
  <VFSearchFormList>
    <input type="submit" value="Access">
  </form>

```

<VFBanner>

This tag is replaced by the contents of the 'Startup Message' box in the Banners tab of the ViewDS Management Agent (see the help topic *View or modify DUA banners*).

This value is stored in the `startup-message` argument of the operational attribute `duaBanners` (see page 86).

If no startup message is defined, the following default is displayed: *Welcome to ViewDS Web edition.*

NOTE: This tag can also be used in three other templates – Search Form, Search Results and Expanded Entry. In each, the tag is replaced by a different message.

<VFVersion>

This tag is replaced by the version of Access Presence that is running.

<VFSearchFormURI [restricted= {user | base | *CHAR}] [id=*CHAR]>

This tag is replaced by the URI for the Search Form selected by the user. It can be used in any Access Presence template.

restricted

The `restricted` argument can be used to set the base entry for the Search Form:

- `user` – the base entry for the search is the entry for the user who is viewing the Search Form.
- `base` – the base entry for the search is the base entry of the DIT.
- `*CHAR` – the base entry is a specific entry identified by either the value of the entry's `entryUUID` attribute; or the LDAP string representation of the entry's Distinguished Name.
- no value – if `restricted` is declared without a value, the base entry for the search will be the current entry.

When `restricted` is not declared, the base object is the default base object for the Search Form. This is the base object defined in the Search Form definition if present; otherwise, it is the base object of the Access Presence set by the configuration-file parameter `baseentry` (see page 16).

id

This argument is common to many tags and is described on page 29.

<VFSearchFormList [showdn] [useScript=1*CHAR]>

This tag is replaced by a drop-down list containing the names of the available Search Forms.

The available Search Forms are declared through either the ViewDS Management Agent (see help topic *View or modify a Search Form*) or the Stream DUA (see *searchOptions* on page 101).

showdn

If `showdn` is specified, the base entry is displayed in LDAP DN string representation before the name of each Search Form in the list.

useScript

This argument invokes a call to a script function when the user selects a Search Form. The script function is identified by the argument's value.

The argument should be within a `<script>` tag with the language set to `text/javascript` (or a scripting type that supports a compatible syntax). Access Presence generates parameter values that are escaped for Javascript string content.

The call has the following format:

```
func(index, name, baseEntry, url)
```

Where:

- `func` – the value of the `useScript` argument.
- `index` – an integer that identifies a Search Form by its position in the sequence of Search Forms defined.
- `name` – a string which is the name of a Search Form.
- `baseEntry` – an LDAP string representation of the DN of the base entry in the Search Form.
- `url` – the URL required to access the Search Form.

`<VFWelcomeURI [id=*CHAR]>`

This tag generates the URI for the Welcome page, and can be used as the value of a `HREF` attribute in an HTML anchor tag.

The `id` argument is common to many tags and is described on page 29.

Search Forms template

This template generates a page that allows users to search the directory using the Search Form they selected on the Welcome page.

The query input fields must be written using the HTML `<FORM>` and `<INPUT>` tags. Both `POST` and `GET` submit methods are supported, although the `POST` method is preferred.

Managing and defining Search Forms

The DSA stores definitions of each Search Form displayed by the Search Forms page. The definitions also describe how the results of a search are sorted and displayed by the Search Results page.

The Search Forms definitions are declared through either:

- ViewDS Management Agent – see the help topic *View or modify a Search Form*
- Stream DUA – see the operational attribute *searchOptions* on page 101

Drop-down lists

The Search Form page can be configured to present the available search attributes in drop-down boxes. It can also be configured to present an attribute's permitted values in a drop-down list.

To present search attributes in a drop-down list:

- ❑ switch `on` the configuration-file parameter `webSelectableField` (see page 19)

To present an attribute's permitted values in a drop-down box:

- ❑ switch `off` the configuration-file parameter `webSelectableField` (see page 19)

- ❑ ensure that the permitted values are defined as the constrained or enumerated attribute syntax or as a result of CompareWords pre-processing function
- ❑ ensure that the attribute is not overloaded – that is, in the ViewDS Management Agent, all values in the ‘column’ box of the Row Attributes window are unique (see the help topic *View or modify a Search Form*).

Attribute ID

When Access Presence starts up, it obtains DUA presentation information from the base entry in the directory. It looks at the first type of `SearchForm` in the `searchOptions` operational attribute and assigns each attribute listed in the `row1-atts`, `row2-atts`, and `row3-atts` attribute lists an ascending number starting from 0. These numbers – each referred to as an *attribute ID* – are used by Access Presence to name each field.

The attributes in each row of a Search Form can be managed through the Stream DUA or ViewDS Management Agent (see the help task *View or modify a Search Form*).

To illustrate how attribute IDs are used, consider the following attributes:

| Attribute Name | ID |
|-----------------------|-----------|
| Surname | 0 |
| Given | 1 |

A following sample query input uses the attribute IDs and `FORM` tag with the `POST` method:

```
<form method="post" action="<VFSearchURI>">
  Surname: <input name="0"><br>
  Given: <input name="1"><br>
  Press <input type="submit" name="<VFDoSearchID>" value="here">
    to submit the query.
</form>
```

Alternatively, a template that supports only one Search Form can identify attributes using their names:

```
<form method="post" action="<VFSearchURI>">
  Surname: <input name="commonName"><br>
  Given: <input name="organizationalUnitName"><br>
  Press <input type="submit" name="<VFDoSearchID>" value="here">
    to submit the query.
</form>
```

Template tags

The Access Presence tags in the template are included in the following example code:

```
<form method="post" action="<VFSearchURI>">
  <VFSearchFields>
    <input type="image" src="../icons/b_show.gif"
      name="<VFDoSearchID>" VALUE="[ Search ]" border=0>
    <br>
    <a href=<VFSearchFormURI><IMG SRC="../icons/b_clear.gif" alt="
      [ Clear ]" border=0></A>
    <br>
```

```

< input type=image src="../../../icons/b_scontx.gif"
name="<VFSetContextID>" VALUE="[ Set Search Context ]" border=0>
<br>
<input type=image src="../../../icons/b_ccontx.gif"
name="<VFClearContextID>" VALUE="[ Clear Search Context ]" border=0>
<br><hr>
<VFSearchOptions>
</form>

```

The above tags, plus others that can appear in this template, are described below.

**<VFSearchURI [restricted= {user | base | *CHAR}]
[id=*CHAR]>**

This tag must be the value of the HTML ACTION tag in a form, and is replaced by the URI for the Search Form page. Alternatively, when it is returned in a form, the tag tells Access Presence to perform a search and output the results.

The `restricted` argument is described on page 31.

The `id` argument is described on page 29.

<VFSearchFieldVal [id=1*DIGIT] [spacing=1*DIGIT]>

This tag is replaced by the display name of the next attribute in the list of attributes in the Search Form. (To view or modify a display name, see the ViewDS Management Agent's help topic *View or modify an attribute's DUA presentation*.) The `id` argument allows an attribute to be identified by either its name or attribute ID (see page 33).

spacing

The `spacing` argument specifies a field width for the display name. If `spacing` is less than the length of the display name, but not zero, Access Presence truncates the display name to fit the specified width.

By default, the display name is right justified – for left justified specify a negative value for `spacing`.

**<VFSearchFields [selectable = { on | off }]
[columns = 1*DIGIT]>**

This tag is replaced by the default search fields defined in the selected Search Form.

selectable

If `selectable` is `on`, the field names for the search attributes are displayed in drop-down list boxes (see page 32) and can be selected by the user. This argument overrides the default behaviour set by the configuration-file parameter `webSelectableField` (see page 19).

columns

This argument specifies the number of columns in which the search fields are displayed. If the argument is unspecified, the default of two columns applies.

<VFSearchOptions>

This tag is replaced by input boxes, drop-down lists and a check-box to allow the user to select the following search options (see *Search Form parameters* on page 19):

- maximum number of entries to return

- maximum time per query
- whether to show subordinates and leaves
- whether to show icons and labels
- whether to show full details if there is a single match

<VFDSearchID>

This tag tells Access Presence to output the name for the submit field that will initiate the search. It should be used to generate the `NAME` parameter value of an `INPUT` field with `TYPE="submit"`.

<VFBanner>

This tag is replaced by the contents of the 'Start Banner 1' box in the Banners tab of the ViewDS Management Agent (see the help topic *View or modify DUA banners*). This value is stored in the `start-banner1` argument of the operational attribute `duaBanners` (see page 86).

NOTE: This tag can also be used in three other templates – Welcome, Search Results and Expanded Entry. In each, the tag is replaced by a different message.

<VFSearchFormName>

This is replaced by the name of the currently selected Search Form.

<VFSearchFormURI>

This tag is replaced by the URI for the Search Form. It can be used in any Access Presence template to provide navigation to the Search Form.

<VFLDAPQueryURI [restricted= {user | base | *CHAR}]>

This tag is replaced by a URI that allows an LDAP filter string to be submitted. The tag should be the value for the `ACTION` attribute in an HTML `FORM`; the `METHOD` attribute of the `FORM` must be set to `POST`.

The `VFLDAPQueryURI` tag should be used in conjunction with `VFLDAPQueryID`.

The `restricted` argument is described on page 31.

<VFLDAPQueryID>

This tag identifies an LDAP filter string. It should be the value of the `NAME` attribute in an HTML `INPUT` tag. The `INPUT` tag should be text field.

The `VFLDAPQueryID` tag should be used in conjunction with `VFLDAPQueryURI`.

Search-context tags

These tags relate to context attributes (see page 8).

To support the context attributes, the Search Form needs to display the existing values stored by the user. (This is done automatically by the default Search Form generated by the `<VFSearchFields>` tag.) The Search Form template uses the following tags that relate to context attributes.

<VFSetContextID>

This is replaced by the name for the submit field that will initiate the set context operation. It should be used to generate the `NAME` parameter value of an `INPUT` field with `TYPE="submit"`.

<VFClearContextID>

This is replaced by the name for the submit field that will initiate the clear-context operation. It should be used to generate the `NAME` parameter value of an `INPUT` field with `TYPE="submit"`.

**<VFSearchField name=1*CHAR [id=1*CHAR]
[html=`1*CHAR`]>**

This tag is replaced by the display name for the search field.

name

If the `name` argument is an attribute type or numeric identifier, the next attribute from the Search Form is displayed.

Alternatively, if the `name` argument is an arbitrary string (neither an attribute-type name nor a numeric identifier), this element generates a list box containing the attributes permitted on the Search Form. In this case, the `id` argument can be used to identify the attribute to be selected by default in the list box. Otherwise, no attribute is selected by default unless the search field is displaying an attribute from the user's search context.

html

The `html` argument allows extra parameters to be declared for the `SELECT` tag that generates the list box.

<VFQueryFieldVal name=1*CHAR [id=1*CHAR] >

This tag is replaced by the stored search-context value for a search field.

NOTE: This tag can also appear with a different syntax in the Search Results template.

name

The `name` argument identifies an attribute type in the search context to be associated with the search field. It is matched with the attribute type of a `VFSearchField` tag.

id

The `id` argument should match the `id` argument of the corresponding `VFSearchField` tag.

Search Results template

The Search Result template generates a page that presents the results of a search to the user. The results are presented in a table, each row containing a directory entry.

The DSA stores Search Form definitions, which also define the sort key for the results on the Search Results page. The sort key is defined through either:

- ViewDS Management Agent – in the Results Sorting area of the Edit Search Form window (see the help topic *View or modify a Search Form*)
- Stream DUA – see the operational attribute `searchOptions` see page 101

The sort algorithm used is set through the configuration-file parameter `websortstrategy` (see page 20).

<VFQueryFieldVal id=1*DIGIT [spacing=1*DIGIT]>

This tag is replaced by the value entered by the user in the Search Form for the search field identified by the `id` argument. The `id` argument identifies an attribute by either its name or attribute ID (see page 33).

The `spacing` argument specifies a field width for the search-field value. If `spacing` is less than the length of the search-field value, but not zero, Access Presence truncates the search-field value to fit the specified width. By default, the value is right justified – for left justified specify a negative value for `spacing`.

NOTE: This tag can only be used in the Search Results template.

<VFSearchNumResult>

This tag is replaced by the number of matching entries returned by the most recent search.

<VFSearchResHeader [format=1*CHAR]>

This tag is replaced by the display names of all requested search fields in the following format:

```
Field1:      Field2:      Field3:.....etc
```

An alternative format can be imposed by setting the `format` element to an identifier declared in the format file (see page 69). To view or modify a display name, see the ViewDS Management Agent's help topic *View or modify an attribute's DUA presentation*.

NOTE: The order in which the fields are displayed is the same as the order of the input fields specified in the Search Form.

**<VFSearchResult [useScript=1*CHAR]
[format=1*CHAR] [alwaysFormat=on|off]>**

This tag is replaced by all matched results in, by default, a tabular format.

If the configuration-file parameter `webShowDetail` (see page 20) is set to `on`:

- for each matched result, the full DN is returned as an HTML reference attached to the result's first attribute.
- each RDN in the DN is formatted as an 'attribute=value' pair, and each RDN is separated by the `&` symbol.
- RDNs are in ascending (LDAP) order.
- the value in the RDN is escaped so that all `"'";,%?+&/:##<>` characters and non-printable characters in the string are mapped to their hex equivalent in the ASCII character set.
- in some cases, where the attribute contains multiple values, each value is escaped as required and separated by the `^` symbol which is encoded as `%2B`.

For example, the name { O "Deltawing"/ OU "Some unit" OU "Asia" } is encoded as:

```
<http://host:port/e.x500/&OU=Some%20unit%2BOU=Asia,&O=Deltawing>
```

useScript

This argument invokes a call to a script function that displays the search results. The script function is identified by the argument's value, and is invoked for each value of each `row1` attribute (of the current Search Form) and for each entry in the search results.

The argument should be within a `<script>` tag with the language set to `text/javascript` (or a scripting type that supports a compatible syntax). Access Presence generates parameter values that are escaped for Javascript string content.

The call has the following format:

```
func(index, objectClass, href, attName, attVal, objDisplayName,
attDisplayName)
```


Where the output depends on the setting of the configuration-file parameter `webShow` (see page 21) and Search Form settings:

- `func` – value of the `useScript` parameter
- `index` – incremental index of entries in the search result 0..n
- `objectClass` – the object class of the search result entry
- `href` – HREF of search result entry
- `attName` – attribute name
- `attVal` – attribute value
- `objDisplayName` – the display name defined for the object class
- `attDisplayName` – the display name defined for the attribute type

alwaysFormat

When this argument is `on`, the formatting information is generated for all `row1` attributes, even when there are no values for the current attribute in the current search result entry. The default behaviour will only generate formatting information when values exist.

By default, this argument is set to `off`.

format

This argument identifies a set of format directives in the format file (see page 69) to be used to generate the Search Result.

<VFQueryFields>

This tag is replaced by a brief summary of the user's input.

For example, if the user enters `sherma` in the surname search field and an `a` in the `givenName` search field, the output will be as follows:

```
surname="sherma" & givenName="a"
```

<VFBanner>

This tag is replaced by the contents of the 'Search Banner' box in the Banners tab of the ViewDS Management Agent (see the help topic *View or modify DUA banners*). This value is stored in the `search-banner` argument of the operational attribute `duaBanners` (see page 86).

NOTE: This tag can also be used in three other templates – Welcome, Search Form and Expanded Entry. In each, the tag is replaced by a different message.

<VFRelatedEntryInput [format=*CHAR]>

During the related-entry workflow (see page 150), Access Presence replaces `VFRelatedEntryInput` with an HTML `INPUT` tag and assigns it a name and value. All attributes declared within the `VFRelatedEntryInput` tag are copied to this replacement `INPUT` tag.

For example, the following tag would be replaced by an `INPUT` tag with an image:

```
<VFRelatedEntryInput type="image" src="../../icons/b_new.gif">
```

The `format` attribute references a set of formatting directives in the format file (see page 69). Only the `StartEntry` and `EndEntry` directives are available for this tag.

<VFRelatedEntryForm> </VFRelatedEntryForm>

This tag block is of use in a Search Results template defined specifically for the related-entry workflow (see page 150).

The related-entry workflow can use either the normal Search Results template or a Search Results template developed specifically for the workflow. The tag block is ignored when it appears in a Search Results template and it is not currently being used in the related-entry workflow. This allows the same Search Results template to be used for the workflow and normal operation.

The tags within the block provide the links required by the workflow to allow the user to add a new role or entry. The links are 'add alternative class' and 'add new entry'.

Examples

The following example results in an Add button being displayed next to each entry in the Search Results page. The Add button invokes the Modify page, which allows the user to associate the alternative class (such as an `organizationalRole`) with an entry.

```
<VFRelatedEntryForm>
  <VFSearchResults>
</VFRelatedEntryForm>
```

NOTE: The above example does not work for Microsoft Internet Explorer (IE). This is because the `VFSearchResults` tag is replaced by an `INPUT` tag with its `TYPE` argument set to `IMAGE`. With IE, the button does not submit a value, but rather the coordinates of the mouse-click on the image.

The following example results in a New button being displayed below the list of entries. This button also invokes the Modify page, but this time allows the user to add a new entry of the same class as the entries listed on the Search Results page.

```
<VFRelatedEntryForm>
  <VFSearchResults>
    New Staff Entry <VFRelatedEntryInput type="image"
      src="../../icons/b_new_entry.gif">
</VFRelatedEntryForm>
```

<VFQueryURI [template=*CHAR] [filterfor=*CHAR] [scope=user|base|*CHAR] [escval=on|off] [id=*CHAR] [pageSize=*DIGIT] [reverse] [raw]>

This tag is replaced by a URL that allows the user to resubmit the search request. The URL reproduces the search request that generated the results currently displayed on the Search Results page.

The tag's arguments are described below.

| | |
|------------------------|--|
| <code>template</code> | Identifies the name of a Search Result template in the extra template file (see page 11). |
| <code>filterfor</code> | Restricts the resubmitted search to a specified space-separated list of object classes. This argument overrides the Search Object Classes field in the Search Form. (See the ViewDS Management Agent help topic <i>View or modify a Search Form</i> .) |

| | |
|-----------------------|---|
| <code>scope</code> | The scope argument changes the scope of the resubmitted search. The scope argument is described on page 29. Unless restricted searches are required, this argument should be set to 'base'. |
| <code>escval</code> | When this argument is <code>on</code> , the value generated by the tag is 'escaped' to make it safe for use as a URI. |
| <code>id</code> | This argument is described on page 29. |
| <code>pageSize</code> | Sets the number of entries displayed on the Search Results page after the user has resubmitted a search request. When this argument is used, the <i>Tags for navigating multiple Search Results</i> (page 41) can be implemented to allow navigation between pages of results. |
| <code>reverse</code> | Indicates that the contents of the Search Results page should be displayed in reverse order when the user clicks the link generated by this tag. |
| <code>raw</code> | Indicates that a resubmitted search request should use the original search criteria, ignoring any modifications made by the remaining <code>VFQueryURI</code> arguments. |

Tags for navigating multiple Search Results

The following tags are replaced by HTML hypertext links that allow users to navigate multiple Search Results pages:

- `VFQueryFirstHref` – replaced by a link to the first page in a set of search results
- `VFQueryLastHref` – replaced by a link to the last page
- `VFQueryNextHref` – replaced by a link to the next page
- `VFQueryPreviousHref` – replaced by a link to the previous page

The tags have the same arguments as the `VFQueryURI` tag (see page 40), except for the `html` argument which is described on page 29.

The number of entries on each page is defined by the `pageSize` argument. If `pageSize` is undefined, then an input field from the Search Form can be used to set the number entries on each page. If the `pageSize` cannot be determined, then the above tags produce no output.

URIs produced

The URIs produced by the above tags encode page information in a query string using four attributes:

- `before_count` – identifies the number of entries to display before the target entry
- `after_count` – identifies the number of entries to display after the target entry
- `content_count` – identifies the number of entries in the set of results
- `target_offset` – identifies the offset of the target entry to display in the result set.
This attribute should normally have a value in the range of 1 to `content_count`.

The `content_count` and `target_offset` attributes are used as a guide. If the actual result set is a different size to the number indicated by `content_count` then

the `target_offset` value is scaled to reference an entry in the same relative position in the result set. Where the result set size is not known, the first page of results may be referenced by setting `target_offset` to 0 and `content_count` to 1 and the last page of results may be referenced by setting the `target_offset` to equal `content_count`.

The `content_count` should always be set to greater than or equal to 1.

Expanded Entry template

The Expanded Entry template generates the page that presents the details of an individual entry. The Expanded Entry includes the full DN of the entry, all associated attributes (unless non-printable or hidden), and a list of the subordinate entries in the Directory Information Tree.

NOTE: The configuration-file parameter `webTemplAttribute` (see page 24) allows alternative Expanded Entry templates to be invoked for specific directory entries. The extra template file (see page 11) allows alternative Expanded Entry templates to be invoked for all users.

The rest of this subsection describes the tags that can be included in the Expanded Entry template:

- Core template tags
- Target object tags
- Alternative hierarchy tags
- Approval process tags

Core template tags

```
<VFExpandDN [useScript=1*CHAR] [reverse=on|off]
[superioronly=on|off] [fromdepth=1*DIGIT]
[format=1*CHAR] [id=*CHAR]>
```

This tag is replaced by the full DN of the entry.

By default, the DN is displayed in rows, each containing one RDN as an 'attribute-value' pair. Each RDN has a HTML reference attached, which a user can click in order to view the RDN's entry.

For example, in plain text the name { C "AU"/ O "Deltawing" } would be displayed as follows:

```
Country AU
Organization Deltawing
```

In HTML it is:

```
Country <a href="http://host:port/view500/webdua.cgi?ea0_1fz99_120.&&c=AU">AU</a>
```

```
Organization <a href="http://host:port/view500/webdua.cgi?ea0_1fz99_120.&&o=Deltawing,c=AU">Deltawing</a>
```

useScript

This argument invokes a call to a script function that displays a DN. The script function is identified by the argument's value, and it is invoked for each component of the DN.

The argument should be within a `<script>` tag with the language set to `text/javascript` (or a scripting type that supports a compatible syntax). Access Presence generates parameter values that are escaped for Javascript string content.

The call has the following format:

```
func(level, objectClass, href, attName, attVal, leaf,
objDisplayName, attDisplayName)
```

Where:

- `func` – value of the `useScript` parameter
- `level` – level of AVA in DN (int 1..n)
- `objectClass` – the object class of the entry described by the complete DN (this is a string and is the same for every AVA)
- `href` – HREF of the DN down to the current level
- `attName` – attribute name
- `attVal` – attribute value
- `leaf` – indicates whether the entry can have subordinates (`True`) or not (`False`)
- `objDisplayName` – the display name defined for the object class
- `attDisplayName` – the display name defined for the attribute type

reverse

This argument generates RDNs in reverse order. Default (`off`) is to start with the entry under the root.

superioronly

When this argument is specified, the last RDN is not included in the generated information.

The default is `off`.

fromdepth

This argument declares an integer depth to begin generating DN information. The RDNs from the root down to depth minus one are ignored.

format

This argument identifies a set of format directives in the format file (see page 69) to be used to generate the DN information.

id

This argument is common to many tags and is described on page 29.

```
<VFExpandAtt [useScript=1*CHAR] [alwaysFormat=on|off]
[id=*CHAR] [format=1*CHAR] [dnformat=1*CHAR] >
```

This tag is replaced by all attributes (except 'hidden' ones) associated with the entry.

By default, the attributes are shown in rows, and long lines are wrapped where necessary. Icons and attribute labels may be attached depending on the configuration-file parameter `webShow` (see page 21).

If the user clicks a non-string attribute, a special 'download' message is displayed and the user can save the attribute to disk. For example, the attributes `telephoneNumber` and `userCertificate` would be displayed as follows:

```
Phone +61 3 9234 5678
User Certificate press here to download
```

useScript

This argument invokes a call to a script function for each attribute in the entry. The argument's value corresponds to the name of the script function.

The argument should be within a `<script>` tag with the language set to `text/javascript` (or a scripting type that supports a compatible syntax). Access Presence generates parameter values that are escaped for Javascript string content.

Access Presence generates a call with the following format:

```
func(objectClass, attName, attVal, objDisplayName,
attDisplayName)
```

Where:

- `func` – value of the `useScript` argument
- `objectClass` – the object class of the entry
- `attName` – attribute name
- `attVal` – attribute value
- `objDisplayName` – the display name defined for the object class
- `attDisplayName` – the display name defined for the attribute type

alwaysFormat

When this argument is `on`, formatting information is generated for all attributes even if the attribute does not have a value in the current entry. When the argument is `off` (the default), formatting information is only generated when a value exists.

id

This argument is common to many tags and is described on page 29.

format

This argument identifies a set of directives in the format file (see page 69) used to display an attribute.

dnformat

This argument identifies a set of directives in the format file (see page 69) used to display a DN.

```
<VFExpandSubclass [useScript=1*CHAR] [ alwaysFormat =
yes|no] [format=1*CHAR] [show=none|nonleaf|all]
[id=*CHAR] [scope= {user | base | *CHAR}]>
```

This tag is replaced by the entries one level below the current entry in the DIT. These subordinate entries are displayed row-by-row and grouped by their object class.

The default appearance of the subordinate entries is controlled through either:

- ViewDS Management Agent – in the Subordinate area of the object class's Properties window (see the help topic *View or modify an object class's DUA presentation*).
- Stream DUA – in the `sub-classes` component of the operational attribute `objectClassPresentation` (see page 94).

The above settings can be overridden by the configuration-file parameter `webshow` (see page 21) and by the search options on the Search Form (see *VFSearchOptions* on page 34).

useScript

This argument invokes a call to a script function that displays a subordinate entry. The script function is identified by the argument's value, and it is invoked for each subordinate entry.

The argument should be within a `<script>` tag with the language set to `text/javascript` (or a scripting type that supports a compatible syntax). Access Presence generates parameter values that are escaped for Javascript string content.

Access Presence generates calls that have the following format:

```
func(index, objectClass, href, attName, attVal, leaf,
objDisplayName, attDisplayName)
```

If `webshow` (see page 21) is set to `sub`, or the equivalent setting is selected on the Search Form, then:

- `func` – value of the `useScript` argument
- `objectClass` – the object class of the subordinate entry
- `href` – HREF of the subordinate entry
- `attName` – attribute name of RDN
- `attVal` – attribute value of RDN
- `leaf` – indicates whether the entry is a branch (`False`) or leaf (`True`)
- `objDisplayName` – the display name defined for the object class
- `attDisplayName` – the display name defined for the attribute type

For any other setting of `webshow`, the above apply, except for `attName` and `attVal`:

- `attName` – attribute name
- `attVal` – attribute value

alwaysFormat

When this argument is `on`, formatting information is generated for all subclasses, even when there are no subordinates for the subclasses in the current entry. When the argument is `off` (the default), formatting information is only generated when values exist.

format

This argument identifies of a set of format directives in the format file (see page 69) used to display subclass information.

show

This argument overrides the current behaviour defining which subordinate entries are displayed. This behaviour is defined by the configuration-file parameter `webshow` (see page 21) and may be modified through the search options on the Search Form.

The argument overrides the current behaviour as follows:

- `none` – do not show any subordinate entries
- `nonleaf` – show only non-leaf subordinate entries
- `all` – show all subordinate entries

id

This argument is common to many tags and is described on page 29.

scope

The `scope` argument is described on page 29.

<VFBanner>

This tag is replaced by the following strings which are concatenated and displayed above the entry's details:

- the *label* declared for the entry's object class (for example, 'Name')
- the entry's `commonName` attribute or, if declared, the object class's *preferred name*
- if declared, the display name and value of the *special attribute* for the entry's object class

The special attribute allows alternative information to be displayed and might be, for example, the entry's telephone number or photograph.

To illustrate, if:

- *label* equals 'Name:'
- *preferred name* is not set
- special attribute is set to `emailAddress` (display name is 'Email')

Then the following would be displayed for an example entry:

Name: "Bill Bailey" Email bill.bailey@cheeseworld.com.au

And if:

- *label* equals 'Staff Number:'
- *preferred name* is set to `employeeNumber`
- *special attribute* is not set

Then the following would be displayed for the same example entry:

Staff Number: "109378777"

NOTE: This tag can also be used in three other templates – Welcome, Search Form, Search Results. In each, the tag is replaced by a different message.

Declaring the label

The label can be declared through either:

- ViewDS Management Agent
Set in the 'Label' box in the DUA Presentation tab of the object class properties window. See the help topic *View or modify an object class's DUA presentation*.
- Stream DUA
Set in the `exp-name-label` component of the operational attribute `objectClassPresentation` (see page 94).

Declaring a preferred name

The preferred name can be declared through either:

- ViewDS Management Agent
Set in the 'Preferred name' box in the DUA Presentation tab of the object class properties window. See the help topic *View or modify an object class's DUA presentation*.
- Stream DUA
Set in the `preferredName` component of the operational attribute `objectClassPresentation` (see page 94).

Declaring the special attribute

The label can be declared through either:

- ViewDS Management Agent
Set in the 'Name' box in the DUA Presentation tab of the object class properties window. See the help topic *View or modify an object class's DUA presentation*.
- Stream DUA
Set in the `special-att` component of the operational attribute `objectClassPresentation` (see page 94).

<VFAssPassFormHref [html=1*CHAR] [id=*CHAR]>

This element is replaced by a hypertext link to the Assign Password page. Access to the Assign Password page is not part of the default Access Presence configuration. To allow access, add this tag to the Expanded Entry template.

The `html` and `id` arguments are common to many tags and are described on page 29.

<VFLabel>

The tag is a simple label and an alternative to the information displayed by the `VFBanner` tag (see page 46).

It is replaced by the value of the mandatory naming attribute (defined by schema) for the current entry. Alternatively, if defined, a 'preferred name' is displayed (see page `preferredName` on page 100).

Target object tags

These tags are replaced by hypertext links to the target-object cache templates described on page 62. For a description of the target-object cache, see page 7.

The configuration-file parameter `webTargetObject` (see page 21) controls whether these links are displayed to all users.

<VFSetTargetObjHref [confirm] [html=1*CHAR [id=*CHAR]>

This tag is replaced by a hypertext link that empties the target-entry cache and then adds the current entry to it.

The `confirm`, `html` and `id` arguments are described on page 29.

**<VFShowTargetObjHref [html=1*CHAR] [id=*CHAR]
[expandSingle= on|off] >**

This tag generates a link that displays the content of the target-object cache. Its behaviour changes according to the number of entries in the target-object cache:

| Target objects | Tag is replaced by... |
|----------------|--|
| None | Hypertext link to the base object |
| One | Hypertext link to the entry set as the target object |
| Multiple | Hypertext link to the Show Target Objects page (see page 63) |

However, if the `expandSingle` argument is present and set to `off`, then the tag generates a link to the Show Target Objects page (see page 63). If the argument is absent or set to `on`, and there is a single entry in the target-object cache, then the tag generates a link to the Expanded Entry page.

The `html` and `id` arguments are described on page 29.

<VFMoveHref [confirm] [html=1*CHAR] [id=*CHAR]>

This tag generates a link that moves the current entry to below a new superior within the directory hierarchy. Its behaviour differs according to the number of entries in the target-object cache:

| Target objects | Replaced by a link that... |
|----------------|--|
| One | Moves the current entry so that it becomes an immediate subordinate of the target object. The <code>confirm</code> argument invokes a confirmation dialog before the entry is moved. |
| Multiple | Opens the Select Target Object page (see page 63), which allows the user to select a new superior for the current entry. |

The `html` and `id` arguments are described on page 29.

<VFImportHref [confirm] [html=1*CHAR] [id=*CHAR]>

This tag generates a link that imports entries from within the directory hierarchy so that they become subordinate to the current entry. Its behaviour differs according to how many entries are in the target-object cache:

| Target objects | Replaced by a link that... |
|----------------|---|
| One | Moves the target object to be an immediate subordinate of the current entry. The <code>confirm</code> argument invokes a confirmation dialog before the entry is moved. |
| Multiple | Opens the Select Entries to Import page (see page 65), which allows the user to select the entries to become subordinates of the current entry. |

The `html` and `id` arguments are described on page 29.

Access Presence only generates a link for this tag if one of the following applies:

- the user has super-user privileges
- `moveMultipleSubs` is enabled through either the ViewDS Management Agent (see help topic *View or modify default entitlements for DUA users*) or through the

Stream DUA in either `defaultEntitlement` (see page 106) or `userEntitlement` (see page 115).

The `html` and `id` arguments are described on page 29

`<VFRemovalsHref [html=1*CHAR] [id=*CHAR]>`

This tag generates a link to the *Select Entries to Remove template* (see page 66) to select and delete multiple entries in the target object cache in a single operation.

The `html` and `id` arguments are described on page 29

Alternative hierarchy tags

These tags relate to the alternative hierarchies (see page 8).

`<VFExpandAltSubord type=AttributeName
[useScript=1*CHAR] [alwaysFormat=yes|no]
[format=1*CHAR] [scope= {user | base | *CHAR}]
[show=none|nonleaf|all] [id=*CHAR] >`

This tag displays an alternative hierarchy of entries below the current entry. It provides an alternative to the standard DIT hierarchy provided by the `VFExpandSubclass` tag.

The alternative hierarchy is based on the attribute identified by the `type` argument. For example, if `type` were set to `manager`, all entries whose `manager` attribute is set to the DN of the current entry would be displayed. This would be irrespective of where they appear in the DIT.

Except for `type`, this tag's arguments are identical to those described under `VFExpandSubclass` on page 44. The `type` argument identifies a DN-type attribute.

`<VFAltAddHref type=AttributeName [confirm]
[html=1*CHAR] [id=*CHAR] >`

This tag generates a link to the *Select Entries to Add to Alternative Hierarchy template* (see page 66).

Access Presence does not always replace the `VFAltAddHref` tag with a link. It will only display a link if:

- the schema and access controls allow the user to add the entries in the target-object cache to the alternative hierarchy
- the entries in the target-object cache are not currently in the alternative hierarchy

The `confirm`, `html` and `id` arguments are described on page 29.

`<VFAltMoveHref type=AttributeName [confirm]
[html=1*CHAR] [id=*CHAR] >`

This tag generates a link to the *Select Entries to Move in Alternative Hierarchy template* (see page 67).

Access Presence will only display the link if:

- the schema and access controls allow the user to move the entries in the target-object cache to the alternative hierarchy
- the entries in the target-object cache are already in the alternative hierarchy

The `confirm`, `html` and `id` arguments are described on page 29.

```
<VFAltRemoveHref type=AttributeName [ confirm ]  
[ html=1*CHAR ] [ id=*CHAR ]>
```

This tag generates a link that removes the current entry from the alternative hierarchy identified by the `VFExpandAltSubord` tag's `type` parameter (see page 49).

Access Presence will only display the link if:

- the user has appropriate permissions to remove the entry
- the current entry is in the alternative hierarchy

The `confirm`, `html` and `id` arguments are described on page 29.

```
<VFAltExpandHref type=AttributeName [ html=1*CHAR ]  
[ id=*CHAR ]>
```

This tag generates a link to display the current entry's subordinates within an alternative hierarchy. Access Presence will only display the link if the current entry has subordinates within the alternative hierarchy identified by the `VFExpandAltSubord` tag's `type` parameter (see page 49).

The `html` and `id` arguments are described on page 29.

Approval process tags

These tags should be added to the Extended Entry template in order to implement the approval process (see page 151). The links and information they generate is only displayed to users with appropriate access rights.

```
<VFRequestDeleteHref [html=1*CHAR] [id=*CHAR]>
```

This tag is replaced by a hypertext link to the Request Remove Entry template (see page 60). The `html` and `id` arguments are described on page 29.

```
<VFRequestAddSubHref [html=1*CHAR] [id=*CHAR]>
```

This tag is replaced by a hypertext link to the Add page, which allows the user to submit a request to add a subordinate to the current entry. The `html` and `id` arguments are described on page 29.

```
<VFRequestModifyHref [html=1*CHAR] [id=*CHAR]>
```

This tag is replaced by a hypertext link to the Modify page, which allows the user to submit a request to modify the current entry. The `html` and `id` arguments are described on page 29.

```
<VFRequestMoveHref [html=1*CHAR] [id=*CHAR]>
```

This tag is replaced by a link that submits a request to move the current entry. The `html` and `id` arguments are described on page 29.

```
<VFRequestImportHref [html=1*CHAR] [id=*CHAR]>
```

This tag is replaced by a link that submits a request to import entries from within the directory hierarchy so that they become subordinate to the current entry. The `html` and `id` arguments are described on page 29.

<VFRequestRemovalsHref [html=1*CHAR] [id=*CHAR]>

This tag generates a link to the *Select Entries to Remove template* (see page 66) to select and request the deletion multiple entries in the target object cache in a single operation.

It is suitable for users that do not have permission to remove any entries in the target object cache but do have permission to submit a request to remove at least one of these entries.

The `html` and `id` arguments are described on page 29

<VFRequestList [id=*CHAR] [caption=*CHAR]>

This tag is replaced by an HTML table of update requests that relate to the current user and are awaiting approval.

The table contains the following information:

- Requestor – a hypertext link to the Expanded Entry page for the user who requested the update.
- Status – the status of the update request.
- Type – the type of update (such as modify, delete or move an entry).
- Time – the timestamp when the request was submitted.
- Reason – the reason for the update entered by the requestor.
- Action – a hypertext link to the page allowing the approver to view and either reject or approve the request.

The `caption` argument can be used to declare the text in an HTML caption in the table generated by this tag. If the argument is undeclared, or is an empty string, then no HTML caption is generated in the table.

The `id` argument is described on page 29.

<VFRequestHistory [id=*CHAR] [caption=*CHAR]>

This tag is replaced by an HTML table containing the list of activities that have occurred on the current update request. It only generates output when the user is reviewing an update request.

The `caption` argument can be used to declare the text in an HTML caption in the table generated by this tag. If the argument is undeclared, or is an empty string, then no HTML caption is generated in the table.

The `id` argument is described on page 29.

Error template

The Error template generates a page that displays ViewDS error messages. The following tags are available in the Error template.

<VFErrorStr [format=1*CHAR]>

This tag is replaced with the unformatted error string if an error has occurred. For the possible error strings, with error numbers, see `VFHasError` below.

The `format` argument is used to identify format file specification (see page 69) that displays the error string in a format appropriate to its context. (The specification should use the key words `StartEntry` and `EndEntry`.)

<VFHasError>

This tag is replaced by a '0' if there is no error present, or by an error number. Each error number has a corresponding error string, which replaces `VLErrorStr` described above.

The error numbers and corresponding error strings are as follows. Several error strings include '%s', which is replaced by appropriate text.

| Error number | Error string |
|--------------|---|
| 12127 | Sorry, at least %s characters are required for the input. |
| 16001 | Sorry, no word match for %s. Please re-enter. |
| 16014 | Cannot connect to the Directory. The Directory or the network may be down. Please try again later. |
| 16016 | The search has identified too many entries. Please be more specific. |
| 16017 | The request has timed out. Please retry later. |
| 16019 | Server busy. Please retry later. |
| 16020 | Sorry, the directory does not have sufficient resources to process this request. |
| 16021 | The entry has been moved or deleted by another user. |
| 16022 | No matching entry could be found. |
| 16024 | A corrupted entry was detected. Please report this to your administrator. |
| 16025 | Your Search/Read has been rejected by the server. |
| 16026 | You do not have sufficient access permission to do this operation. |
| 16027 | The entry has been modified by another user. |
| 16028 | The Directory does not accept duplicate values. |
| 16029 | The entry already exists. |
| 16030 | The operation is not allowed on non leaf node. |
| 16037 | Sorry, please give a name to the new entry. |
| 16048 | A system level error has been detected. Please report this to your administrator. |
| 16049 | Unexpected Error. Please report the following number to your administrator: |
| 16126 | Sorry, an invalid/non-printable character has been detected in field %s. Please remove the character and retry. |
| 16128 | Sorry, field %s is too long. |
| 16129 | Sorry, only single value is allowed for %s. |
| 16130 | Sorry, a non unique value is detected in field %s. Please remove any non-unique value in the field. |
| 16131 | Sorry, the phone number %s entered is invalid. Please re-enter. |
| 16132 | Sorry, please fill in the mandatory field %s . |

| Error number | Error string |
|--------------|---|
| 16133 | Sorry, the time you entered in field %s is invalid. Please re-enter the time. |
| 16134 | Sorry, your input for field %s is of an invalid format. Please re-enter the field. |
| 16135 | Sorry, the OR Address you entered is invalid. Please re-enter. |
| 16151 | Your Username and/or Password is incorrect. Please check them before trying to login again. |

Modify template

The Modify template generates a page that allows a user to modify an entry. It also allows a user to enter the details for a new entry, and ensures that the values available for selection conform to schema and access controls.

The following tags are only outputted if the user has the required privileges.

<VFModyfHref [confirm] [html=1*CHAR] [id=*CHAR]>

This tag is replaced by a hypertext link to the Modify page, which allows the user to modify the current entry.

The `confirm`, `html` and `id` arguments are described on page 29.

<VFModifyForm [id=*CHAR] name=1*CHAR [summary=1*CHAR]>

This tag block is replaced by an HTML FORM element and should contain the `VFModifyAtt` tag.

For example:

```
<VFModifyForm id="modform" name="modform" summary="Modify">
  <table>
    <caption><h4>Modify details for<VFLabel></h4></caption>
    <VFModifyAtt width="36"><VLErrorStr>
  </table>
</VFModifyForm>
```

The `id` argument is described on page 29.

<VFModifyAtt [width=1*DIGIT] [id=*CHAR] [format=*CHAR] [description={row | heading | data}] [save = {top | bottom | both}] [checkbox[=1*CHAR]]>

This tag is replaced by modifiable input controls containing an entry's attribute values. It should be included in the code block `<VFModifyForm> </VFModifyForm>`.

An attribute with a Distinguished Name syntax is displayed in a list box unless the optional `checkbox` argument is included (see below):

- a single-valued attribute is displayed in a single-select list box
- a multi-valued attribute is displayed in a multiple-select list box

In both cases, the user can select an empty option. The content of the list box is sorted alphabetically according to the values generated by the `VFLabel` tag (see page 47).

The entries in the target-object cache (see page 7) are presented in a single-select list box.

The `format` and `id` arguments are described on page 29. The `width` argument specifies the width of the input boxes for attribute values (the default is 50).

save

This argument controls whether the Save button is displayed above, below, or both above and below the form's data. The default is `top`.

description

This argument controls which part of a row responds to the user's mouse hovering over it. The response is to display the description defined in the attribute's DUA presentation – see the ViewDS Management Agent help topic *View or modify an attribute's DUA presentation*.

The `description` argument can have one of the following values:

- `row` – the entire row (the default) responds to the mouse
- `heading` – the heading cell of the row responds to the mouse
- `data` – the input field responds to the mouse

checkbox

This argument indicates that a multi-valued attribute is displayed using checkbox input fields, rather than a multiple-select list box. This occurs only for multi-valued attributes whose permitted values are well-defined, either through a constrained syntax, enumerated type or the use of pre-processing functions. It also occurs for the auxiliary object class list.

The checkbox input fields are generated for each possible value of the attribute. These are all contained within a FIELDSET HTML element. The `checkbox` has an optional value. If this value is provided, it will be used as a value for the class attribute of the FIELDSET HTML elements generated on the Modify form.

<VFDeleteHref [confirm] [html=1*CHAR] [id=*CHAR]>

This tag is replaced by a hypertext link that deletes the current entry. The `confirm` argument invokes a confirmation dialog when the link is clicked by a user.

The `html` and `id` arguments are common to many tags and are described on page 29.

Modify Value Form template

The Modify Value Form template generates a page that allows a user to add, modify and delete individual values of a multi-value attribute. It also ensures that the values available for selection conform to schema and access controls.

The following tags are only outputted if the user has the required privileges.

<VFModifyForm [id=*CHAR] name=1*CHAR [summary=1*CHAR]>

This tag block is replaced by an HTML FORM element and should contain the `VFModifyVal` tag.

For example:


```
<VFormModifyForm id="modform" name="modform" summary="Modify">
  <table>
    <caption><h4>Modify details for<VFormLabel></h4></caption>
    <VFormModifyVal width="36"><VFormErrorStr>
  </table>
</VFormModifyForm>
```

The `id` argument is described on page 29.

**<VFormModifyVal [width=1*DIGIT] [id=*CHAR] [format=*CHAR]
[description={row | heading | data}] [save = {top | bottom
| both}] [checkbox[=1*CHAR]]>**

This tag is replaced by modifiable input controls containing values. It should be included in the code block `<VFormModifyForm> </VFormModifyForm>`.

An attribute with a Distinguished Name syntax is displayed in a multiple-select list box unless the optional `checkbox` argument is included (see below):

The user can select an empty option. The content of the list box is sorted alphabetically according to the values generated by the `VFormLabel` tag (see page 47).

The entries in the target-object cache (see page 7) are presented in a single-select list box.

The `format` and `id` arguments are described on page 29. The `width` argument specifies the width of the input boxes for attribute values (the default is 50).

save

This argument controls whether the Save button is displayed above, below, or both above and below the form's data. The default is `top`.

description

This argument controls which part of a row responds to the user's mouse hovering over it. The response is to display the description defined in the attribute's DUA presentation – see the ViewDS Management Agent help topic *View or modify an attribute's DUA presentation*.

The `description` argument can have one of the following values:

- `row` – the entire row (the default) responds to the mouse
- `heading` – the heading cell of the row responds to the mouse
- `data` – the input field responds to the mouse

checkbox

This argument indicates that a multi-valued attribute is displayed using checkbox input fields, rather than a multiple-select list box. This occurs only for multi-valued attributes whose permitted values are well-defined, either through a constrained syntax, enumerated type or the use of pre-processing functions. It also occurs for the auxiliary object class list.

The checkbox input fields are generated for each possible value of the attribute. These are all contained within a `FIELDSET` HTML element. The `checkbox` has an optional value. If this value is provided, it will be used as a value for the class attribute of the `FIELDSET` HTML elements generated on the Modify Value form.

<VFAddValueInput [html=1*CHAR]>

This tag must be used within the scope of the HTML form created using the `VFModifyForm` tag.

It will generate a submit button on the Modify Value form suitable for adding a new value if the current user has permission to add a value for the attribute.

This tag produces an HTML input control with both type and name attributes. If an optional "html" parameter is present, the remainder of the control, including the closing '>' for the element, are provided by this parameter. Otherwise, the value and title attributes are provided with default values.

<VFModifyValueInput [html=1*CHAR]>

This tag must be used within the scope of the HTML form created using the `VFModifyForm` tag.

It will generate a submit button on the Modify Value form to modify the current value if the current user has permission to modify the value of the attribute.

This tag produces an HTML input control with both type and name attributes. If an optional "html" parameter is present, the remainder of the control, including the closing '>' for the element, are provided by this parameter. Otherwise, the value and title attributes are provided with default values.

<VFDeleteValueInput [html=1*CHAR]>

This tag must be used within the scope of the HTML form created using the `VFModifyForm` tag.

It will generate a submit button on the Modify Value form to delete the current value if the current user has permission to delete the value of the attribute.

This tag produces an HTML input control with both type and name attributes. If an optional "html" parameter is present, the remainder of the control, including the closing '>' for the element, are provided by this parameter. Otherwise, the value and title attributes are provided with default values.

<VFPreprocessInput [html=1*CHAR]>

This tag must be used within the scope of the HTML form created using the `VFModifyForm` tag.

It will generate a checkbox input control that can be used to enable/disable optional preprocessing for a value of an attribute. The input control will only be generated when preprocessing is applicable for the value. Preprocessing cannot be applied to attributes with complex syntax, such as annotated types.

This tag produces an HTML input control with both type and name attributes. If an optional "html" parameter is present, the remainder of the control, including the closing '>' for the element, are provided by this parameter. Otherwise, the id, value and title attributes are provided with default values.

Add template

The Add template generates a page that allows the user to select the object class for a new entry.

<VFAddSubHref [confirm] [html=1*CHAR] [id=*CHAR]>

This tag is replaced by a hypertext link to the Add page. The `confirm`, `html` and `id` arguments are common to many tags and are described on page 29.

<VFAddOCSel>

This tag is replaced by a list box containing all the available entry types (object classes) that the user can create under the current entry. It should be used within a `FORM` block which has an `ACTION` parameter value generated by `VFAddOCURI` (see below).

<VFAddOCURI [id=*CHAR]>

This tag is replaced by a URI pointing to the Modify page from the Add page. It should be used to generate the `ACTION` parameter value for a `FORM` containing the `VFAddOCSel` tag.

The `id` argument is common to many tags and is described on page 29.

Print Form template

The Print Form template generates a page that allows the user to request a report. The list of available reports is configured through the `printconfig` file (see *Configuring for printing* on page 25).

A requested report is generated by the Printing DUA (see page 125), and the results are displayed according to the MIME mapping file (see page 13). If no MIME type is set for the report in the `printconfig` file then the report will be displayed by the Print template (see page 58).

<VFPrintFormHref [html=1*CHAR] [id=*CHAR]>

This tag is replaced by a hypertext link to the Print Form page. The `html` and `id` arguments are common to many tags and are described on page 29.

<VFPrintSelection [width=1*DIGIT] [useScript=1*CHAR] [scope= {user | base | *CHAR}]>

This tag is replaced by the dialog that allows a user to select a report to be printed.

The `width` argument specifies the width of the input fields in the selection dialog.

The input arguments are:

- The *print report type*, a list box of the print scripts available to the user, as defined in the file `printconfig` (see *Configuring for printing* on page 25).
- The *base entry* from which to generate the report. The presents a drop-down list containing the entries in the target-object cache (see *Target-object* on page 62). If the user does not select an entry, the report's base entry will be the one defined in the print script.
- Optionally, the *output file name*. If this is not provided, the report will only be displayed to the screen.

This tag can be used in any other template to display the list of reports with hypertext links to each. When used in a template other than the Print Form, however, it can only be invoked using the `useScript` argument. If used in the Welcome template, the `scope` argument must be set to `base`.

useScript

This argument generates a call to a script function identified by the argument's value.

The argument should be within a `<script>` tag with the language set to `text/javascript` (or a scripting type that supports a compatible syntax). Access Presence generates parameter values that are escaped for Javascript string content.

The call has the following format:

```
func(id, label, base, entry, url)
```

Where:

- `func` – the value of the `useScript` argument
- `id` – an integer that identifies a print script by its position in the file `printconfig`
- `label` – a string that identifies a print script by its name in the file `printconfig`
- `base` – the LDAP string encoding for the DN of the base entry from which the report will be generated
- `url` – the URL required to access the report

scope

This argument allows you to change the scope of a report. By default the scope of a report is the current entry and its subordinates. The `scope` argument is described on page 29.

Print template

The Print template generates a page to present the report requested through the Print Form page.

<VFPrintReport >

This tag is replaced by the generated report.

New Password template

The New Password template allows a user to change their password – the user enters their current password and then a new password twice.

<VFChangePassFormHref [html=1*CHAR] [id=*CHAR]>

This tag is replaced by a hypertext link to the New Password page. The `html` and `id` arguments are common to many tags and are described on page 29.

<VFChangePassURI [id=*CHAR]>

This tag is used to construct a URI to submit the New Password page. It should be used to generate the `ACTION` parameter value for a `FORM` containing the `VFAuthUserPassID`, `VFAuthUserPassID` and `VFAuthUserPassID2` tags.

```
<FORM METHOD="post" ACTION="<VFChangePassURI">>
```

The `id` argument is common to many tags and is described on page 29.

<VFAuthUserPassID>

This tag is the value of the `name` field in the HTML `input` tag for the user's current password:

```
<input name="VFAuthUserPassID" type="password" size="10">
```

The `type` field should be set to `password` so that the user's password is not displayed on the screen.

<VFAuthUserPassID>

This tag is the value of the `name` field in the HTML `input` tag for the user's new password:

```
<input name="VFAuthUserPassID" type="password" size="10">
```

The `type` field should be set to `password` so that the user's password is not displayed on the screen.

<VFAuthUserPassID2>

This tag is the value of the `name` field in the HTML `input` tag for the user's confirmed password:

```
<input name="VFAuthUserPassID2" type="password" size="10">
```

The `type` field should be set to `password` so that the user's password is not displayed on the screen.

<VFAuthReferer>

This tag ensures that Access Presence returns to the page the user was attempting to view before resetting their password.

The tag is replaced by an HTML input element:

```
<input type='hidden' name='referrer' value=URL>
```

The `value` is equal to the URL for the page the user was attempting to view.

Assign Password template

The Assign Password template allows a user to assign or change the user name and password of another user. It is subject to the access controls that apply to the user attempting to access the page.

NOTE: To provide access to the Assign Password page, a link to the page must be added to the Expanded Entry template (see <VFAssPassFormHref> on page 47).

The `VFAuthUserPassID` and `VFAuthUserPassID2` fields are described under the New Password template on page 58.

<VFAssPassTable>

This tag is replaced by an HTML table containing all the input fields required for the Assign Password page. The fields include the current user name and password.

The tag must be within an HTML `form` element that has an action provided by the `VFAssPassURI` template `template` tag. Any text included between the `VFAssPassTable` tag's name and closing `>` is included as an attribute of the generated HTML table element's start tag.

```
<form method="post" action="<VFAssPassURI>">
  <VFAssPassTable>
    <input type="submit" name="submit" value="Change Password">
  </form>
```

<VFAssPassURI [id=*CHAR]>

This tag is replaced by a URI to submit the Assign Password form. It is used to generate the value for the ACTION element in a FORM containing the VFAssPassTable, VFAuthUserNameID, VFAuthUserPassID and VFAuthUserPassID2 tags.

```
<form method="post" action="<VFAssPassURI>">
```

The id argument is common to many tags and is described on page 29.

<VFAuthUserNameID>

This tag is the value of the name field in the HTML input tag for the user name of the user whose user name or password are to be changed:

```
<input name="<VFAuthUserNameID>" value="<VFAttVal id=userName>"
type="text">
```

The type field should be set to text.

Request Remove Entry template

This template is part of the approval process (see page 151).

The Request Remove Entry page is invoked by the link generated by the VFRequestDeleteHref tag (see page 50). For a 'requestor', the page allows the user to enter the reason why they want to delete the current user; and for an 'approver', the page allows the user to view the reason and either reject or approve the deletion.

The Request Remove Entry template should include the following tags enclosed within an HTML FORM element:

- VFRequestReason (see page 82)
- VFRequestSaveInput (see page 82)
- VFRequestCancelInput (see page 82)
- VFRequestApproveInput (see page 83)
- VFRequestRejectInput (see page 83)

The method attribute of the FORM should be set to post and its action attribute set to the output of the VFRequestDeleteURI template file tag (see below).

<VFRequestDeleteURI [html=1*CHAR] [id=*CHAR]>

This tag generates the value of the action attribute in an HTML FORM in the Request Remove Entry template. The html and id arguments are described on page 29.

Global changes templates

These templates allow an Access Presence user to modify attributes globally. To illustrate, if a user changes an attribute's value globally, then all instances of the attribute (within the scope of the change) will be set to the new value.

Before a user can apply global changes, the ViewDS DSA must be configured to enable this functionality (see *Configuring for global changes* on page 26).

There are three global changes templates:

- Global Change Request template
- Global Change Confirm template
- Global Change Results template

Each is described below.

Global Change Request template

The Global Change Request template presents a form that allows a user to specify a global change.

<VFGCRequestHref html=*CHAR>

This tag provides a link to the Global Change Request page. For example:

```
<VFGCRequestHref html=">Global Changes">
```

The `html` argument is described under *Common arguments* on page 29.

<VFGCRequestURI [id=*CHAR] [html=1*CHAR]>

This tag must be included in the template, and is replaced by a URI to submit the Global Change Request page and display the Global Change Confirm page. It generates the value for the `ACTION` element in a `FORM` containing the `VFGCRequest` tag. The `FORM` should have its `METHOD` set to `POST`, and should include a submit button. For example:

```
<form method="post" action="<VFGCRequestURI>">
    ...
    <VFGCRequest width="36">
    <VFErrorStr format="error">
    <input type="image" src="../icons/newface/save.jpg">
    ...
</form>
```

The `id` and `html` arguments are described under *Common arguments* on page 29.

<VFGCRequest [width=1*DIGIT]>

This tag is replaced by form elements that allow the user to specify a global change. The user can select the scope of the change, the entry type and attribute to apply the change to, the nature of the change (add, modify, replace value), the new value, etc.

The `width` argument specifies the length of the input boxes displayed.

Global Change Confirm

The Global Change Confirm template generates a page showing a summary of the global changes defined in the Global Change Request.

<VFGCConfirmURI [id=*CHAR] [html=1*CHAR]>

This tag must be included in the template, and is replaced by a URI to submit the Global Change Confirm page and display the Global Change Results page. It generates the value for the `ACTION` element in a `FORM` containing the `VFGCConfirm` tag. The `FORM` should have its `METHOD` set to `POST`, and should include a submit button.

For example:

```
<form method="post" action="<VFGCConfirmURI>">
```

```

...
<VFGCConfirm width="32">
<VFEErrorStr format="error">
<input type="image" src="../icons/newface/save.jpg">
...
</form>

```

The `id` and `html` arguments are described under *Common arguments* on page 29.

<VFGCConfirm [width=1*DIGIT]>

This tag is replaced by a list of the entries, and their attributes, that will be affected by the global changes defined in the Global Change Request. It also displays check boxes to allow the user to confirm each change before they are applied.

The `width` argument specifies the length of the input boxes displayed.

Global Change Results

The Global Change Results template generates a page that shows whether a global change was successful for each entry it was applied to.

<VFGCResults [id=*CHAR]>

This tag must be included in the template, and is replaced by a list of entries and the status of the global change for each.

The `id` argument is described under *Common arguments* on page 29.

Target-object cache templates

For an overview of the target-object cache, see page 7.

This subsection describes the tags and templates relating to the target-object cache. The tags that are common to many templates are described below; and the tags that can appear in templates and the format file are described on page 80.

Common target object tags

These tags only apply to the first entry in the target-object cache. They can appear in any Access Presence template except the Authentication template.

<VFTargetObjStr>

This tag is replaced by a string representation of the DN of the current 'target object'. If no target object is set, nothing is generated.

<VFTargetObjDN [useScript=1*CHAR]>

This tag is replaced by the LDAP DN of the current target object, if there is one set. This replacement string will be escaped for HTML, with quotes and angle brackets replaced by their character entity reference representations (for example, & is replaced by `"`).

useScript

This argument generates a call to a script function identified by the argument's value.

The argument should be within a `<script>` tag with the language set to `text/javascript` (or a scripting type that supports a compatible syntax). Access Presence generates parameter values that are escaped for Javascript string content.

The call has the following format:

```
func(ldapDN, objectClass)
```

Where:

- `ldapDN` – a quoted string containing the LDAP DN of the current target object, escaped appropriately for use as a script string
- `objectClass` – a string representing the object class of the current target object, if available; otherwise an empty string

<VFTargetObjOC>

Deprecated.

Show Target Objects template

This template allows a user to view the contents of, and delete entries from, the target-object cache.

The Show Target Objects page is displayed when a user:

- clicks the link generated by `VFShowTargetObjHref` tag (see page 48)
- submits a form utilising the `VFShowTargetObjectsURI` tag described below

<VFShowTargetObjectsURI [id=*CHAR] >

This tag must be included in the template, and is replaced by a URI that submits the Show Target Object page. It generates the value for the `ACTION` element of a `FORM` that uses the `POST` method and contains the `VFShowTargetObjects` tag. For example:

```
<form method="post" action="<VFShowTargetObjectsURI>">
  <table width="100%">
    <tr>
      <th><input type="checkbox" name="toggle-all" value="true"
        onchange="toggle_all(this);"></th>
      <th>Name</th>
      <th>Unit</th>
      <th>Telephone</th>
    </tr>
    <VFShowTargetObjects format="targets">
    <VFErrorStr format="error">
  </table>
  <input type="image" src="../icons/newface/delete.jpg">
</form>
```

The `id` argument is described under *Common arguments* on page 29.

<VFShowTargetObjects [format=1*CHAR] [id=*CHAR] >

This tag is replaced by a list of entries in the target-object cache. They are sorted alphabetically according to the values generated by the `VFLabel` tag (see page 47).

The `VFShowTargetObjects` tag must be within an HTML FORM whose ACTION element is set to the `VFShowTargetObjectsURI`.

The `VFShowTargetObjects` tag must also include the `format` parameter to identify a format-file specification which includes the `VFSelectTargetObjInput` tag (see page 81). The `VFSelectTargetObjInput` tag is replaced by HTML INPUT controls (checkboxes) that allow the user to select the entries to be removed from the target-object cache. They are removed when the FORM is submitted.

To illustrate, consider the following `VFShowTargetObjects` tag:

```
<VFShowTargetObjects format="targets">
```

Its `format` parameter identifies the following format-file specification:

```
<format * * targets>
<tr>
  <td><VFSelectTargetObjInput type="checkbox"></td>
  <td><a href="<VFURI"><VFLabel></a></td>
  <td><VFAttVal id="hierarchyName"></td>
  <td><VFAttVal id="telephoneNumber"></td>
</tr>
</format>
```

The `id` argument is described under *Common arguments* on page 29.

Select Target Object template

This template displays the entries in the target-object cache that are valid potential superiors of the entry being moved within the directory hierarchy. It allows the user to select the new superior.

The Select Target Object page is displayed when a user clicks the link generated by `VFMoveHref` tag (see page 48). That is, if there is more than one entry in the target-object cache that is a valid superior for the entry being moved.

<VFSelectTargetObjURI [id=*CHAR] >

This tag must be included in the template, and is replaced by a URI that submits the Select Target Object page. It generates the value for the ACTION element of a FORM that uses the POST method and contains the `VFSelectTargetObj` tag. For example:

```
<form method="post" action="<VFSelectTargetObjURI">>
  <table width="100%">
    <tr>
      <th>Name</th>
      <th>Unit</th>
      <th>Telephone</th>
    </tr>
    <VFSelectTargetObj format="select"> <VLErrorStr>
  </table>
  <input type="image" src="../icons/newface/move.jpg">
</form>
```

The `id` argument is described under *Common arguments* on page 29.

`<VFSelectTargetObj [format=1*CHAR] [id=*CHAR] >`

This tag is replaced by a list of entries in the target-object cache that are valid superiors to the entry being moved. They are sorted alphabetically according to the values generated by the `VFLabel` tag (see page 47).

The `VFSelectTargetObj` tag must be within an HTML FORM whose ACTION element is set to the `VFSelectTargetObjURI`.

The `VFSelectTargetObj` tag must also include the `format` parameter to identify a format-file specification which includes the `VFSelectTargetObjInput` tag (see page 81). The `VFSelectTargetObjInput` tag is replaced by HTML INPUT controls (radio buttons) that allow the user to select the entry to be the new superior to the entry being moved. To illustrate, consider the following `VFSelectTargetObj` tag:

```
<VFSelectTargetObj format="select">
```

Its `format` parameter identifies the following format-file specification:

```
<format * * select>
<tr>
  <td><VFSelectTargetObjInput type="radio"></td>
  <td><a href="<VFURI>"><VFLabel></a></td>
  <td><VFAttVal id="hierarchyName"></td>
  <td><VFAttVal id="telephoneNumber"></td>
</tr>
</format>
```

The `id` argument is described under *Common arguments* on page 29.

Select Entries to Import template

This template displays the entries in the target-object cache that are valid subordinates to the current entry. It allows the user to select new subordinates for the current entry.

The Select Entries to Import page is displayed when a user clicks the link generated by the `VFImportHref` tag (see page 48).

`<VFSelectImportsURI [id=*CHAR] >`

This tag must be included in the template, and is replaced by a URI that submits the Select Entries to Import page. It generates the value for the ACTION element of a FORM that uses the POST method and contains the `VFSelectImports` tag.

The `id` argument is described under *Common arguments* on page 29.

`<VFSelectImports [format=1*CHAR] [id=*CHAR] >`

This tag is replaced by a list of entries in the target-object cache that are valid subordinates to the current entry. They are sorted alphabetically according to the values generated by the `VFLabel` tag (see page 47).

The `VFSelectImports` tag must be within an HTML FORM whose ACTION element is set to the `VFSelectImportsURI`.

The `VFSelectImports` tag must also include the `format` parameter to identify a format-file specification which includes the `VFSelectTargetObjInput` tag (see page 81). The `VFSelectTargetObjInput` tag is replaced by HTML INPUT controls (checkboxes) that allow the user to select the new subordinate entries. The `id` argument is described under *Common arguments* on page 29.

Select Entries to Remove template

This template displays the entries in the target-object cache that are valid candidates to be removed. It allows the user to select and delete or select and request the deletion of multiple entries in a single operation, depending on their permissions.

The Select Entries to Remove page is displayed when a user clicks the link generated by the `VFRemovalsHref` tag (see page 49) or the `VFRequestRemovalsHref` tag (see page 51).

`<VFSelectRemovalsURI [id=*CHAR] >`

This tag must be included in the template, and is replaced by a URI that submits the Select Entries to Remove page. It generates the value for the `ACTION` element of a `FORM` that uses the `POST` method and contains the `VFSelectRemovals` tag.

The `id` argument is described under *Common arguments* on page 29.

`<VFSelectRemovals [format=1*CHAR] [id=*CHAR] >`

This tag is replaced by a list of entries in the target-object cache that are valid candidates for removal. They are sorted alphabetically according to the values generated by the `VFLabel` tag (see page 47).

The `VFSelectRemovals` tag must be within an `HTML FORM` whose `ACTION` element is set to the `VFSelectRemovalsURI`.

The `VFSelectRemovals` tag may also include the `format` parameter to identify a format-file specification describing how each entry in this list should be displayed. This format-file specification must include the `VFSelectTargetObjInput` tag (see page 81). The `VFSelectTargetObjInput` tag is replaced by `HTML INPUT` control that is used to identify which entries should be removed when the enclosing `HTML FORM` is submitted.

The `id` argument is described under *Common arguments* on page 29.

Select Entries to Add to Alternative Hierarchy template

This template allows a user to add entries to an alternative hierarchy. The user selects the entries from the target-object cache to become new subordinates (within the alternative hierarchy) of the current entry.

The Select Entries to Add to Alternative Hierarchy page displays the entries in the target-object cache that are:

- valid possible subordinates – within an alternative hierarchy – to the current entry
- are NOT currently in the alternative hierarchy

The *current entry* is the entry displayed by the Expanded Entry page from which this page is invoked by the user – see the `VFAltAddHref` tag on page 49. The *alternative hierarchy* is identified by the `type` parameter of the `VFExpandAltSubord` tag (see page 49).

`<VFAltAddSelectionURI [id=*CHAR] >`

This tag must be included in the template, and is replaced by a URI that submits the Select Entries to Add to Alternative Hierarchy page. It generates the value for the `ACTION` element of a `FORM` that uses the `POST` method and contains the `VFAltAddSelection` tag.

The `id` argument is described under *Common arguments* on page 29.

<VFAltAddSelection [format=1*CHAR] [id=*CHAR] >

This template displays the entries in the target-object cache that are:

- valid possible subordinates – within an alternative hierarchy – to the current entry
- are NOT currently in the alternative hierarchy

They are sorted alphabetically according to the values generated by the `VFLabel` tag (see page 47).

The `VFAltAddSelection` tag must be within an HTML FORM whose ACTION element is set to the `VFAltAddSelectionURI`.

The `VFAltAddSelection` tag must also include the `format` parameter to identify a format-file specification which includes the `VFSelectTargetObjInput` tag (see page 81). The `VFSelectTargetObjInput` tag is replaced by HTML INPUT controls (checkboxes) that allow the user to select the new subordinate entries.

The `id` argument is described under *Common arguments* on page 29.

Select Entries to Move in Alternative Hierarchy template

This template allows a user to move entries within an alternative hierarchy. The user selects the entries from the target-object cache to become new subordinates (within the alternative hierarchy) of the current entry.

The Select Entries to Move in Alternative Hierarchy page displays the entries in the target-object cache that are:

- valid possible subordinates – within an alternative hierarchy – to the current entry
- are already in the alternative hierarchy

The *current entry* is the entry displayed by the Expanded Entry page from which this page is invoked by the user – see the `VFAltMoveHref` tag on page 49. The *alternative hierarchy* is identified by the `type` parameter of the `VFExpandAltSubord` tag (see page 49).

<VFAltMoveSelectionURI [id=*CHAR] >

This tag must be included in the template, and is replaced by a URI that submits the Select Entries to Move in Alternative Hierarchy page. It generates the value for the ACTION element of a FORM that uses the POST method and contains the `VFAltMoveSelection` tag.

The `id` argument is described under *Common arguments* on page 29.

<VFAltMoveSelection [format=1*CHAR] [id=*CHAR] >

This tag is replaced by a list of entries in the target-object cache that are:

- valid possible subordinates – within an alternative hierarchy – to the current entry
- are already in the alternative hierarchy

They are sorted alphabetically according to the values generated by the `VFLabel` tag (see page 47).

The `VFAltMoveSelection` tag must be within an HTML FORM whose ACTION element is set to the `VFAltMoveSelectionURI`.

The `VFAltMoveSelection` tag must also include the `format` parameter to identify a format-file specification which includes the `VFSelectTargetObjInput` tag (see page 81). The `VFSelectTargetObjInput` tag is replaced by HTML INPUT controls (checkboxes) that allow the user to select the new subordinate entries.

The `id` argument is described under *Common arguments* on page 29.

Chapter 5

Format file

A format file contains Access Presence tags that define the presentation of individual elements within pages. It allows a finer level of presentation detail to be defined than is possible in a page template.

This chapter has the following sections:

- Location and syntax
- Assessment order and examples
- URI and link tags
- Display name tags
- Target object tags
- Alternative hierarchy tags
- Approval process tags

Location and syntax

The format file is identified by the configuration-file parameter `webFormatFile` (see page 25). An example format file is supplied with the demonstration directory, Deltawing (`webdir\templ\extra.lst`).

The file contains directives in the following form:

```
<format attName objClass formatID optional>

    <html to replace the attribute with the attName or
    objClass goes here along with the format file tags
    described in this chapter>

</format>
```

The elements of a directive are described below.

attName

The name of an attribute type, or `*` to match any attribute type.

When the `dnformat` argument is declared, the `attName` should be the attribute holding the entry's DN. (The optional `dnformat` is an argument for the `VFExpandAtt` and `VFAttVal` tags.)

objClass

The name of an object class, or `*` to match any object class.

formatID

This is an identifier that corresponds to the value of a tag's `format` argument.

Many of the template tags have the optional `format` argument. When declared, the `format` argument's value will correspond to blocks of format-file entries identified by `formatID`. The directives within these entries are then applied to the data generated by the tag.

optional

A format directive can include one of the following optional keywords.

| Optional keyword | Displays the output defined in the format directive... |
|------------------------------|---|
| <code>StartEntry</code> | Before each entry in the search results or in a subordinate list, if it is present. |
| <code>EndEntry</code> | After each entry in the search results or in a subordinate list, if it is present. |
| <code>StartFirstEntry</code> | Before the first entry in a search result or subordinate list, if it is present. If not present, <code>StartEntry</code> will be used. This keyword applies to the first entry on the page. So for a paged search result, the first entry on each page will be formatted using the <code>StartFirstEntry</code> context. |
| <code>EndFirstEntry</code> | After the first entry in a search result or subordinate list, if it is present. If not present, <code>EndEntry</code> will be used. This keyword applies to the first entry on the page. So for a paged search result, the first entry on each page will be formatted using the <code>EndFirstEntry</code> context. |
| <code>FirstAtt</code> | To the first attribute in each entry. |
| <code>StartLine</code> | Before the first value of each attribute type. |
| <code>ClassName</code> | After the output defined in the <code>StartEntry</code> format directive, and before each entry. This keyword is used to generate the display name for subordinate entries displayed by the Expanded Entry page. |
| <code>StartAtt</code> | Before the first value of a multi-value DN type attribute. |
| <code>EndAtt</code> | After the last value of a multi-value DN type attribute. |
| <code>NoAtt</code> | This keyword allows content to be provided for attributes with no value (for example to provide content for a table cell when there is no value to put in it). |
| <code>RelatedEntryRow</code> | Before each entry displayed by the Search Results page when used in the related-entry workflow (see page 150). |
| <code>RelatedEntryEnd</code> | After the last entry displayed by the Search Results page when used in the related-entry workflow (see page 150). |
| <code>RequestReason</code> | This keyword relates to the approval process (see page 151) and is described below. |
| <code>InputChecked</code> | This keyword relates to the approval process (see page 151) and is described below. |

2. The user selects one of the valid superiors; and because the approval process is implemented, they also complete the 'request reason' box and submit their request. The request is now pending approval.
3. Another user with appropriate access rights logs on (an approver) and views the pending request. As above, it is displayed by the Select Target Object template.

This is the point where InputChecked keyword is used. It defines the presentation so that the template displays the set of valid superiors with the requestor's choice selected. Hence, the approver can then accept or reject the request.

Example:

```
<format * * select InputChecked>
<tr>
  <td><VFSelectTargetObjInput type="radio" checked></td>
  <td><a href="<VFURI>"><VFLabel></a></td>
  <td><VFAttVal id="hierarchyName"></td>
  <td><VFAttVal id="telephoneNumber"></td>
</tr>
</format>
```

Assessment order and examples

The order in which format directives are assessed is as follows:

```
<format      attName      objClass      formatID>
<format      attName      *              formatID>
<format      *            objClass      formatID>
<format      *            *              formatID>
<format      attName      *              *      >
<format      *            objClass      *      >
<format      *            *              *      >
```

A directive with an optional keyword will, however, take precedence over all the above if it is appropriate.

To illustrate, consider two format directives:

```
<format      surName      orgPerson      Details>
    < . . tags . .>
</format>

<format      *            *              Details  FirstAtt>
    < . . tags . .>
</format>
```

If the first attribute were `surName`, the second directive would take precedence. Otherwise, it would be ignored and the first directive would be processed.

Example 1

The format directives in this example produce an alternative layout for the Search Results page. Rather than being in a table, entries are list items. Figure 3 shows the Search Results page containing the results of a search on the surname 'Smith'.

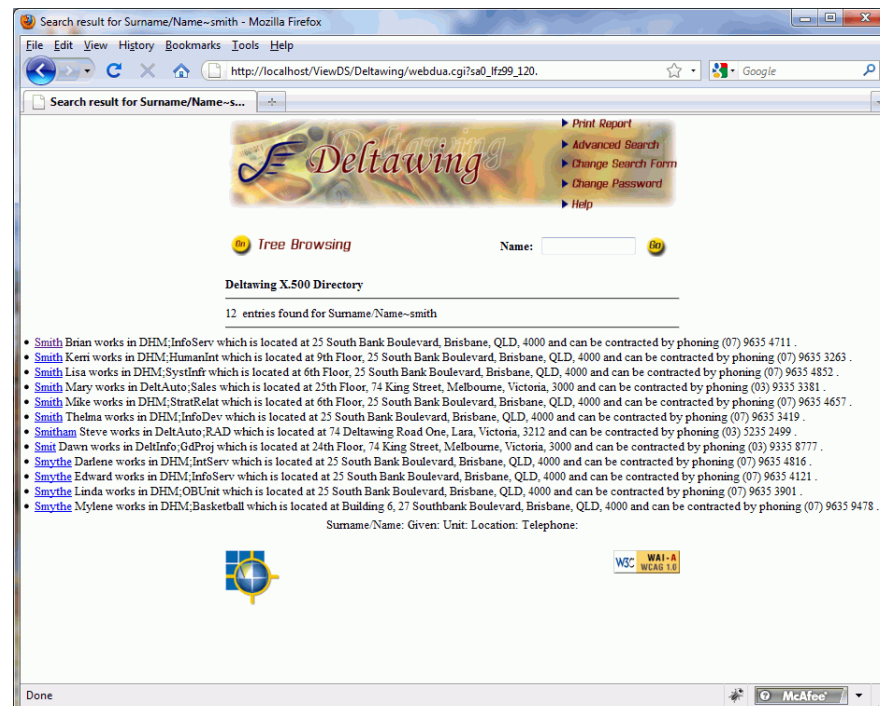


Figure 3: Alternative Search Results format

The set of directives is included in the format file (`webdir\tmpl\extra.lst`) supplied with the demonstration directory, Deltawing, and has been reproduced below. For the sake of the example, each directive has been numbered.

```

<format * * searchR StartEntry>          - 1
    <LI>
</format>
<format * * searchR EndEntry>            - 2
    </LI>
</format>
<format * * searchR FirstAtt>            - 3
    <VFHref><VFAttVal></A>
</format>
<format location * searchR>              - 4
    which is located at <VFAttVal>
</format>
<format organizationalUnitName organizationalUnit searchR FirstAtt> - 5
    Unit <VFHref><VFAttVal></A>
</format>
<format hierarchyName organizationalPerson searchR>          - 6
    works in <VFAttVal>
</format>
<format organizationalUnitName organizationalUnit searchR>    - 7
    Unit <VFAttVal>
</format>
<format telephoneNumber organizationalPerson searchR>        - 8
    and can be contacted by phoning <VFAttVal>
</format>
<format telephoneNumber organizationalUnit searchR>          - 9
    has a common telephone number: <VFAttVal>
</format>
<format * * searchR>          - 10
    <VFAttVal>
</format>

```

For the results shown in Figure 3, the directives are processed as follows (the numbers on the left correspond to those in the above example):

- 1 The HTML tag `` is generated.
- 3 The Access Presence tag `<VFHref>` is replaced with the entry's first attribute and linked to the entry's Expanded Entry page. The `<VFAttVal>` tag is replaced by the next attribute value in the entry, first name.
- 6 The text 'works in' is displayed followed by the next attribute value.
- 4 The text 'which is located at' is displayed followed by the next attribute value.
- 8 The text 'and can be contacted by phoning' is displayed followed by the next attribute value.
- 2 The HTML tag `` is generated.

Example 2

This example comprises two format file directives:

```
<format manager * parentEntry>
  <VFAttVal id=manager DNFormat=nameAndTitle>
</format>

<format * * nameAndTitle>
  <div class="entryName"><VFHRef><VFAttVal id=commonName></a></div>
</format>
```

The first directive relates to the attribute `manager`, and tells Access Presence to apply the `<VFAttVal>` tag (see page 78) to the attribute. The tag's `DNFormat` argument invokes the second directive, which tells Access Presence to display the manager's `commonName` as a hypertext link to their entry. (The `DNFormat` argument is only relevant to DN type attributes.)

URI and link tags

These tags relate to URIs and links, and are primarily intended for use in the format file, but may also be used in the Access Presence templates.

<VFURI [id=*CHAR] [scope= {base | entry | *CHAR}]>

This tag is replaced by the URI for the current entry. The `id` argument is described on page 29; and the `scope` argument is described on page 29.

<VFHRef [id=*CHAR] [html=1*CHAR]>

This tag is replaced by a hypertext link to the Expanded Entry page displaying the current entry.

The `id` argument can be used to create a hypertext link to an extra Expanded Entry template (see page 11). Set this argument to `'_default'` to cause the standard Expanded Entry template to be used.

The `html` argument is described on page 29.

<VFAddValueHref [id=*CHAR] [html=1*CHAR]>

This tag is replaced by a hypertext link to the Modify Value Form for adding a new value to a multi-value attribute, if the current user has permission to add values of the attribute.

The `id` and `html` arguments are described on page 29.

NOTE: This tag can only be used to generate links on the Expanded Entry page.

<VFModifyValueHref [id=*CHAR] [html=1*CHAR]>

This tag is replaced by a hypertext link to the Modify Value Form for modifying or removing an existing value of a multi-value attribute, if the current user has permission to modify or remove values of the attribute.

The `id` and `html` arguments are described on page 29.

NOTE: This tag can only be used to generate links on the Expanded Entry page.

**<VFOOrderHref action={ top | bottom |up | down }
[id=*CHAR] [html=1*CHAR]>**

This tag is replaced by a hypertext link that allows a user to change the position of the selected entry within its list of sibling entries of the same structural object class.

The `action` attribute has one of the following values:

- `top` – moves the entry to the top of the group
- `bottom` – moves the entry to the bottom of the group
- `up` – moves the entry up one position in the group
- `down` – moves the entry down one position in the group

The `id` and `html` arguments are described on page 29.

NOTE: This tag can also appear in an Expanded Entry template.

<VFOResetHref html=*CHAR>

This tag is replaced by a hypertext link that allows a user to move the selected entry to its default position within its set of sibling entries of the same structural object class.

The default location is based on the alphabetical order of the entry's first attribute.

The ability to move entries is provided to users by implementing the `VFOOrderHref` tag (see page 76.)

NOTE: This tag can also appear in an Expanded Entry template.

<VFRandomNum>

This tag causes Access Presence to generate a random number. It is used to generate a value for the `id` attribute of an HTML tag.

<VFOEL [style=crlf]>

This tag is replaced by an end-of-line character sequence. The character sequence generated is platform dependent – for example, on Unix `LF` and on Windows `CRLF`.

When the `style` argument is set to `crlf`, the tag generates the `CRLF` sequence as the end-of-line character sequence.

Display name tags

Object classes and attributes have display names (and optional icons) which are displayed by Access Presence.

An attribute has a default display name. It can also have a different display name (and icon) to be displayed when the attribute is associated with a specific object class.

For example, consider an attribute called `name`, which is used by two object classes: `country` and `device`. The display name for the attribute when it is used by the `country` object class might be 'Country Name'; and the display name when it is used by the `device` object class might be 'Device Name'. When Access Presence displays these objects, it displays the appropriate display name for the attribute.

An attribute display name (and icon) is defined through either:

- ViewDS Management Agent – see the help topic *View or modify an attribute's DUA presentation*.

- Stream DUA – see the `display-name` component of the operational attribute `attributePresentation` on page 87.

An object class display name is declared through either:

- ViewDS Management Agent – see the help topic *View or modify an object class's DUA presentation*.
- Stream DUA – see the `displayName` component of the operational attribute `objectClassPresentation` on page 94.

The following tags can also appear in any Access Presence template.

```
<VFAttVal id=1*CHAR [escval=on|off] [list=1*DIGIT]
[width=1*DIGIT [preChar=CHAR] [postChar=CHAR]]
[scope= {base | user | *CHAR}] [unqualified]
[DNFormat=1*CHAR] [delimiter=1*CHAR]
[valueformat=1*CHAR] [[reference=1*CHAR] |
[path=1*CHAR]] >
```

This tag is replaced by the set of values for the attribute named by the `id` argument. The values are for the attribute in the currently displayed entry.

| | |
|---|---|
| <code>escval</code> | When the <code>escval</code> argument is <code>on</code> , the value generated by this tag is 'escaped' to make it safe for use as a URI. |
| <code>list</code> | <p>This argument can be used to reference a specific value in a multi-valued attribute. The first value is identified by a 1:</p> <pre>List=1</pre> <p>Access Presence ignores this argument if it is set to zero, a negative number, or a non-numeric character.</p> <p>When <code>list</code> is absent, all values are displayed, separated by the delimiter defined for in the attribute's DUA presentation. See the ViewDS help topic <i>View or modify an attribute's DUA presentation</i>, or <code>attributePresentation</code> on page 87.</p> |
| <code>width</code> | Specifies the maximum field width for attribute values. If an attribute value is longer than the <code>width</code> , it is truncated and appended with the characters <code>..</code> . |
| <code>preChar</code> <code>postChar</code> | <p>The <code>preChar</code> argument specifies a padding character to precede values shorter than the field width set by the <code>width</code> argument.</p> <p>The <code>postChar</code> argument specifies a padding character to append to values whose length is shorter than the field width set by the <code>width</code> argument.</p> <p>When neither <code>preChar</code> nor <code>postChar</code> are defined, <code>postChar</code> defaults to a space character. When both are defined, short values are centred in the field width defined by <code>width</code>.</p> |
| <code>scope</code> | The <code>scope</code> argument is described on page 29. |
| <code>unqualified</code> | Affects the behaviour of Access Presence when a user searches on a component of an attribute with a complex syntax (such as certificates and XML documents). If the argument is declared, Access Presence replaces the tag with all component values; otherwise, Access Presence replaces the tag with just the component value the user searched on. |
| <code>dnFormat</code> | References a format file directive that defines how a DN type attribute is displayed. It allows more than just the name of an entry to be displayed to the user – see <i>Example 2</i> on page 75. |

| | |
|-------------|---|
| delimiter | <p>Identifies a delimiter character that overrides the default set for an attribute (see the ViewDS Management Agent help topic <i>View or modify an attribute's DUA presentation</i>).</p> <p>For example:</p> <pre><VFAttVal id='businessCategory' delimiter=';'></pre> |
| valueformat | <p>Identifies the format file specification for the individual values of a multi-valued attribute. The referenced format file specification is restricted to the following tags and arguments (all of which are described in detail elsewhere in this guide):</p> <ul style="list-style-type: none"> • VFAttName - all supported arguments allowed • VFAttIcon - all supported arguments allowed • VFAttType – all supported arguments allowed • VFAttVal – <code>escval</code>, <code>width</code>, <code>preChar</code>, <code>postChar</code>, <code>dnformat</code>, <code>reference</code> or <code>path</code> • VFQueryURI – <code>id</code>, <code>escval</code>, <code>raw</code>, <code>scope</code> <p>The referenced format specification can also include optional format directives (see page 70). For example, the directive <code>FirstValue</code> allows the first value of a multi-valued attribute to be formatted differently to subsequent values.</p> |
| reference | <p>Specifies an ASN.1 component reference to identify a simple component of a complex ASN.1 type to be displayed. The simple components supported are Booleans, integers, strings, object identifiers, date-time fields and distinguished names. This argument only applies when the VFAttVal tag is used to format individual values of a multi-valued attribute. Only one out of <code>reference</code> and <code>path</code> may be included. If the <code>reference</code> argument is present then any <code>path</code> argument will be ignored.</p> |
| path | <p>Specifies an XML component path to identify a simple component of a complex XML type to be displayed. The simple components supported are Booleans, integers, strings, object identifiers, date-time fields and distinguished names. The argument only applies when the VFAttVal tag is used to format individual values of a multi-valued attribute. Only one out of <code>reference</code> and <code>path</code> may be included. If the <code>reference</code> argument is present then any <code>path</code> argument will be ignored.</p> |

```
<VFAttName id=1*CHAR [ifChanged] [force]
[scope={ base | user | *CHAR} ]>
```

This tag is replaced by the display name for the attribute named by the `id` argument.

`ifChanged` If the `ifChanged` argument is declared, the display name is only displayed if it has changed since the tag was last invoked.

`force` The `force` argument overrides the behaviour of the '*show icons only*' option presented in the search options on the Search Form. This can be useful for the search result headings.

`scope`

The `scope` argument is described on page 29.

<VFAttIcon id=1*CHAR [scope={ base | user | *CHAR}]>

This tag is replaced by the icon associated with the attribute named by the `id` argument. The icon must be a `.gif` file stored in the `../icons` directory.

The `scope` argument is described on page 29.

<VFAttType id=1*CHAR [scope={ base | user | *CHAR}]>

This tag is replaced by the type of the attribute named by the `id` argument.

The `scope` argument is described on page 29.

<VFClassName [ifChanged]>

This tag is replaced by the display name of the object class of the current entry. This can be the entry in the Expanded Entry page or the subentry in a list of subordinate entries.

If the `ifChanged` argument is declared, the display name is only displayed if it has changed since the tag was last invoked.

**<VFAttSize id=1*CHAR [scope= {base | user | *CHAR}]
[unqualified] [label=*CHAR] [modifier=1*DIT]>**

This tag is replaced by the number of values in a multi-valued attribute named by the `id` argument.

`scope`

The `scope` argument is described on page 29.

`unqualified`

Affects the behaviour of Access Presence when a user searches on a component of an attribute with a complex syntax (such as certificates and XML documents). If the argument is declared, Access Presence replaces the tag with all component values; otherwise, Access Presence replaces the tag with just the component value the user searched on.

`label`

Allows a label to be set to prefix the number of attribute values. The label is only displayed if the number of values in the attribute is greater than 1.

This argument is intended to provide support for the HTML `rowspan` and `colspan` properties in the HTML table row and cell elements.

For example, `<tr<VFAttSize label=" rowspan">>` results in `<tr>` for a single-value attribute, and `<tr rowspan="2">` for an attribute with two values.

`modifier`

A signed integer modifier that is applied to the number of attribute values. Useful, for example, to generate meaningful “rowcount” values.

Target object tags

These tags relate to the target object (see page 7) and can appear in either a target object template (see page 62) or format file.

<VFAddTargetObjHref [confirm] [html=1*CHAR [id=*CHAR]]>

This tag is replaced by a link that adds the current entry to the target-object cache. Access Presence, however, will only generate the link if the entry is not in the target-object cache. The `confirm`, `html` and `id` arguments are described on page 29.

<VFDeleteTargetObjHref [confirm] [html=1*CHAR [id=*CHAR]]>

This tag is replaced by a link that removes the current entry from the target-object cache. Access Presence, however, will only generate the link if the entry is not in the target-object cache. The `confirm`, `html` and `id` arguments are described on page 29.

<VFSelectTargetObjInput>

This tag generates HTML `INPUT` controls, each with a `NAME` and `VALUE`. The `INPUT` controls must be submitted in a `FORM` that uses the `POST` method and whose `ACTION` is the URL generated by the `VFShowTargetObjectsURI` tag:

```
<form method="post" action="<VFShowTargetObjectsURI">">
```

Each target object template (see page 62) includes a tag which identifies a format-file specification containing the `VFSelectTargetObjInput` tag. The result is that `INPUT` controls are displayed that allow the user to select a target object in the cache. The behaviour when the form is submitted differs according to each template.

Any parameters for the `VFSelectTargetObjInput` tag (aside from `NAME` and `VALUE`) should be declared, including a `TYPE` parameter to ensure the correct type of element is generated. For example:

```
<VFSelectTargetObjInput type="checkbox">
```

The `TYPE` should be 'checkbox' for pages that allow multiple items to be selected, and 'radio' for pages that allow only a single item to be selected.

The `VFSelectTargetObjInput` tag can also appear in a format-file specification referenced by several other template tags: `VFExpandDn`, `VFExpandSubclass`, `VFExpandAltSubord` and `VFSearchResults`. This allows entries in a subordinate listing or search result – which have been selected by the user – to be added to the target-object cache.

Alternative hierarchy tags

These tags relate to the target object (see page 7) and can be used in a format file to list subordinate entries or search results:

- `VFAltRemoveHref` (see page 50)
- `VFAltExpandHref` (see page 50)
- `VFLabel` (see page 47)

The `VFAltType` tag described below can also be used in a format file or template.

**<VFAltType [type=AttributeName] [force]
[scope={ base | user | *CHAR }]>**

This tag can be used in a format file and the following templates:

- Select Entries to Add to Alternative Hierarchy template (see page 66)
- Select Entries to Move in Alternative Hierarchy template (see page 67)

It generates the label for the alternative hierarchy identified by the `VFExpandAltSubord` tag's `type` parameter (see page 49). This can be overridden by the `type` attribute of the `VFAltType`.

The `force` argument overrides the behaviour of the '*show icons only*' option presented in the search options on the Search Form. This can be useful for the search result headings.

The `scope` argument is described on page 29.

Approval process tags

The tags in this subsection can be used in templates and format files.

They can be used in the format file to define an alternative presentation of the input elements displayed on the Modify page during the approval process (see page 151).

The tags should appear in a format-file directive that includes the optional keyword `RequestReason` (see page 70). The directive can then be invoked by the `format` argument of the `VFModifyAtt` tag (see page 53).

These tags are also used in the Request Remove Entry template (see page 60).

<VFRequestReason [rows=1*DIGIT] [cols=1*DIGIT]

This tag is replaced by an HTML `INPUT` text box that allows the user to enter a comment when submitting or modifying an update request.

The tag can be included in a format file entry, which has the keyword `RequestReason`, in order to change the appearance of the input box. (Note that Access Presence automatically adds a default input box to the pages that allow an update request to be submitted or modified.)

The `rows` and `cols` arguments set the dimensions of the text box.

<VFRequestSaveInput [html=1*CHAR] >

This tag is replaced by an HTML `INPUT` element (by default, a submit button) that allows the user to submit an update request.

The tag can be included in a format file entry, which has the keyword `RequestReason`, in order to change the appearance of the input element. (Note that Access Presence automatically adds a default submit button to a page that allows a request to be submitted.)

The default value of the `html` argument (see page 29) is as follows:

```
'value="Submit" title="Submit" type="submit">'
```

The HTML input element is only generated if the user has permission to submit the update request.

<VFRequestCancelInput [html=1*CHAR]>

This tag is replaced by an HTML `INPUT` element (by default, a submit button) that allows the user to cancel an update request.

The tag can be included in a format file entry, which has the keyword `RequestReason`, in order to change the appearance of the input element. (Note that Access Presence automatically adds a default button to a page that allows a request to be cancelled.)

The default value of the `html` argument (see page 29) is as follows:

```
'value="Revoke" title="Revoke" type="submit">'
```

The HTML input element is only generated if the user has permission to cancel the update request.

<VFRequestApproveInput [html=1*CHAR]>

This tag is replaced by an HTML `INPUT` element (by default, a submit button) that allows the user to approve an update request.

The tag can be included in a format file entry, which has the keyword `RequestReason`, in order to change the appearance of the input element. (Note that Access Presence automatically adds a default button to a page that allows a request to be approved.)

The default value of the `html` argument (see page 29) is as follows:

```
'value="Approve" title="Approve" type="submit">'
```

The HTML input element is only generated if the user has permission to approve the update request.

<VFRequestRejectInput [html=1*CHAR]>

This tag is replaced by an HTML `INPUT` element (by default, a submit button) that allows the user to reject an update request.

The tag can be included in a format file entry, which has the keyword `RequestReason`, in order to change the appearance of the input element. (Note that Access Presence automatically adds a default button to a page that allows a request to be rejected.)

The default value of the `html` argument (see page 29) is as follows:

```
'value="Reject" title="Reject" type="submit">'
```

The HTML input element is only generated if the user has permission to reject the update request.

Chapter 6

Server-side attributes

This chapter describes the attributes stored by the Directory System Agent (DSA) that control different aspects of Access Presence. For example, there are presentation operational attributes that specify how a particular class of entry is displayed by Access Presence.

This chapter has the following sections:

- Important note
- Concepts
- DUA presentation operational attributes
- User operational attributes
- Approval process operation attributes
- New entry operation attributes
- Other operational attributes

Any DUA can use the attributes described in this chapter. Access Presence uses all the attributes unless otherwise stated in an individual attribute description.

Important note

The operational attributes described in this chapter can be managed using the Stream DUA (see the *Technical Reference Guide: Directory System Agent*).

Alternatively, you can manage the same attributes through the ViewDS Management Agent. The application has a help system that includes an overview of the concepts and functionality described in this chapter.

NOTE: See the help topic *Key concepts for DUA presentation*.

Concepts

When a DUA starts up, it binds to the DSA and obtains *user-specific operational attributes*. The DUA then determines the base entry, which allows it to obtain schema and *presentation operational attributes*.

Each of these steps is described below.

Obtain user-specific operational attributes

After binding to a DSA, the DUA obtains user-specific operational attributes from the user's entry in the Directory Information Tree (DIT).

The attributes are:

- `userEntitlement`
- `userConfig`
- `privilege`

These attributes are normally changed automatically by a running DUA, or by an administrator performing administration functions with an interactive DUA. The Stream DUA is not normally used with these attributes.

Obtain a base entry

The DUA attempts to obtain the Distinguished Name (DN) of a base entry from one of the following in the order shown:

- the user-specific operational attribute, `userConfig`
- the subschema administrative point that governs the user's entry
- the configuration-file parameter `baseentry` (when Access Presence and the DSA are on different hosts, each has its own configuration file; when located on the same host, Access Presence and the DSA share the same configuration file)

Obtain operational attributes

The DUA obtains schema and presentation operational attributes from the base entry. They provide the DUA with its understanding of the DSA's schema, and configure its start-up banners and messages, the layout of the Search and Expanded Entry page, operational limits, and other features.

Schema operational attributes

The DUA obtains the following schema operational attributes (described in the *Technical Reference Guide: Directory System Agent*):

- `attributeTypes`
- `objectClasses`
- `matchingRuleUse`
- `nameForms`
- `dITStructureRules`
- `dITContentRules`
- `definitions`

They define the schema from the base entry, and are usually declared as part of configuring schema rather than the DUA.

DUA presentation operational attributes

These attributes define the appearance of different aspects of the DUA. They can be modified using the Stream DUA, or the ViewDS Management Agent, and must be in a subschema subentry. For information about the Stream DUA, see the *Technical Reference Guide: Directory System Agent*.

The DUA presentation operational attributes are described in the following subsection.

DUA presentation operational attributes

These attributes are single-valued, except for `attributePresentation` and `objectClassPresentation` which are multi-valued:

- `duaBanners`
- `attributePresentation`
- `objectClassPresentation`
- `searchOptions`
- `defaultEntitlement`

`duaBanners`

The `duaBanners` operational attribute specifies:

- information displayed by the DUA at logon (or soon after)
- the banner for the Search Form
- an optional set of key mappings to override the built-in mappings (deprecated)

It is defined as follows:

```
duaBanners ATTRIBUTE ::= {
    WITH SYNTAX          DUABanners
    SINGLE VALUE          TRUE
    USAGE                 directoryOperation
    ID                    {vf 18 4} }

DUABanners ::= SEQUENCE {
    start-banner1          [0] TeletexString OPTIONAL,
    start-banner2          [1] TeletexString OPTIONAL,
    startup-message        [2] SEQUENCE OF TeletexString OPTIONAL,
    search-banner          [3] TeletexString OPTIONAL,
    vt100-key-overrides    [4] KeyMappings OPTIONAL,
    pc-key-overrides       [5] KeyMappings OPTIONAL }

KeyMappings ::= SEQUENCE OF KeyMapping -- DEPRECATED
KeyMapping ::= SEQUENCE { -- DEPRECATED
    action                [0] DUAAction,
    keys                   [1] SEQUENCE OF Key }

DUAAction ::= ENUMERATED {
    startfield, endfield, -- etc -- refresh }

Key ::= ENUMERATED {
    kb-a, kb-b, -- etc -- alt-fn12 }
```

Components

start-banner1

`start-banner1` specifies a string displayed on the first line of the Search Form page of Access Presence.

For example: `start-banner1 "DELTAWING CORPORATE DIRECTORY"`

start-banner2

Deprecated.

startup-message

`startup-message` specifies a string displayed on the first line of the Welcome page of Access Presence.

search-banner

`search-banner` specifies a string displayed at the top of the Search Results page of Access Presence.

For example:

```
search-banner "DELTAWING CORPORATE DIRECTORY"
```

Examples

The following Stream DUA script reads the `duaBanners` attribute. Because DUA presentation attributes are available throughout a subschema area, the attributes can be read from the Deltawing entry itself.

```
read
    organizationName "Deltawing"
return duaBanners;
```

The following script sets `duaBanners`. As the DUA presentation attributes can only be modified at the subentry where they are held, the modify operation must be directed to the subschema subentry.

```
modify
    organizationName "Deltawing"
    / commonName "Subschema"
with changes {
    remove attribute duaBanners,
    add attribute duaBanners {
        start-banner1 "DELTAWING CORPORATE DIRECTORY",
        start-banner2 "On-line Edition",
        startup-message {
            "Caution!!", "System going down at 3.30. p.m."
        },
        search-banner "DELTAWING CORPORATE DIRECTORY"
    }
};
```

attributePresentation

The multi-valued `attributePresentation` attribute specifies how attributes are displayed in the Search, Expanded Entry and Modify Value Form pages. It has the following ASN.1 definition:

```
attributePresentation ATTRIBUTE ::= {
    WITH SYNTAX          AttributePresentation
    EQUALITY MATCHING RULE objectIdentifierFirstComponentMatch
    USAGE                directoryOperation
    ID                   {vf 18 5} }
AttributePresentation ::= SEQUENCE {
    type                  [0] ATTRIBUTE.&id,
    display-names         [1] SEQUENCE OF DisplayName,
    flags                 [2] AttributeFlags OPTIONAL,
    delimiters            [3] TeletexString DEFAULT ";" ,
```

```

        preprocessing      [4] SEQUENCE OF PrepFunction OPTIONAL,
        attrib-hiding      [5] SEQUENCE OF AttribHiding OPTIONAL,
        replace-strings    [6] ReplaceStrings OPTIONAL,
        description        [7] UnboundedDirectoryString OPTIONAL,
        dateTimeFormat      [8] UTF8String OPTIONAL,
        fields              [9] AttributeFields OPTIONAL )

DisplayName ::= SEQUENCE {
    name          TeletexString,
    classes       SEQUENCE OF OBJECT-CLASS.&id OPTIONAL }
AttributeFlags ::= BIT STRING {
    fixedWidth      (0),          -- Not Used
    globallyChangeable (1),
    matchingValuesOnly (2),
    approximateMatch (3),          -- Not Used
    formattedText   (4),
    isEmailAddress  (5),
    isURL           (6),
    isHTML          (7),
    isBinary        (8),
    sortDNValues    (9) }

AttribHiding ::= SEQUENCE {
    object-class    OBJECT-CLASS.&id,
    privileges      SET OF AccessLevel }
AccessLevel ::= ENUMERATED {
    none           (0),
    read           (1),
    update         (2),
    admin          (3),
    superuser      (4) }

ReplaceStrings ::= SEQUENCE OF ReplaceString
ReplaceString ::= SEQUENCE {
    replace        (0) TeletexString ,
    with           (1) TeletexString }

AttributeFields ::= SEQUENCE OF AttributeField
AttributeField ::= SEQUENCE {
    name           (0) UTF8String , -- field name in syntax
    label          (1) UTF8String , -- display name for field
    behaviour AttributeFieldBehaviour DEFAULT normal:NULL }

AttributeFieldBehaviour ::= CHOICE {
    normal         (2) NULL ,
    formatted      (3) NULL ,
    constrained    (4) SEQUENCE of UTF8String ,
    uuid           (5) NULL ,
    uri            (6) NULL }

```

For the ASN.1 definition of PrepFunction, see *Preprocessing functions* on page 109.

Components

type

`type` specifies the attribute to which the value of `attributePresentation` applies. It is either a symbolic name or object identifier. For example:

```
type surname
type {2 5 4 4}
```

display-names

`display-names` specifies a sequence of *display name records*. Each consists of:


















- a string which is the *display name* of the attribute; and
- a sequence of structural object classes to which display name applies.








When the attribute is displayed in the Expanded Entry page, the DUA displays the name that corresponds to the structural object class of the entry.

The first display-name record should have an omitted or empty sequence of object class. It is the *default name* used – for example, on the Search Form or for a structural object class that is not listed explicitly. Usually, this default name is sufficient and further display-name records are unnecessary.

The structural object classes can be specified by name or object identifier.

A `display-name` string can be appended with a '%' character followed by an integer. The integer identifies an icon displayed next to (or instead of) the display name. The icons and their numbers are shown below.

| Icon No | Icon | Application |
|---------|---|----------------------------|
| 10 |  | Locality |
| 20 |  | Generic non-leaf entry |
| 21 |  | Org unit entry |
| 22 |  | Org unit entry |
| 23 |  | (No suggested application) |
| 24 |  | (No suggested application) |
| 30 |  | Generic leaf entry |
| 31 |  | Person entry |
| 32 |  | Person entry |
| 33 |  | Group |
| 40 |  | Telephone no. |
| 41 |  | Telephone no. |
| 50 |  | Facsimile no. |
| 51 |  | Device entry |
| 60 |  | Postal address |
| 61 |  | Email address |
| 62 |  | X.400 address |

| Icon No | Icon | Application |
|---------|---|-----------------|
| 70 |  | Description |
| 71 |  | Description |
| 80 |  | Role / See also |
| 81 |  | Role / See also |
| 82 |  | Note |
| 90 |  | Note |
| 91 |  | Note |

For example:

```
display-names {
    {name "Telephone%41"},
    {name "Enquiries%41", classes {organizationalUnit}},
    {name "Phone%41", classes {conferenceRoom}} }
```

In this example:

- for the object class `organizationalUnit`, the display name of the attribute type is `Enquiries`
- for object class `conferenceRoom`, the display name of the attribute type is `Phone`
- for all other object classes, the display name of the attribute type is `Telephone`

In all cases, the DUA displays icon 41 next to the attribute.

flags

`flags` specifies a set of flags for an attribute type:

| | |
|---------------------------------|---|
| <code>globallyChangeable</code> | The attribute is available for global changes. (See <i>Global changes templates</i> on page 60.) |
| <code>matchingValuesOnly</code> | If the attribute has multiple values, this flag specifies that the Search Results page will only list the value that matches the search criteria. |
| <code>formattedText</code> | Specifies that the DUA should display the attribute's value in a text box. A user can enter a value with simple text formatting (line-breaks and tabs), which the DUA will retain. |
| <code>isEmailAddress</code> | Specifies that the attribute's value should be displayed as an email address. For example, with the value <code>name@host</code> , the DUA would display: <code>name@host</code> |
| <code>isURL</code> | Specifies that the attribute's value is a URL (Universal Reference Locator; see <i>RFC 1738</i>). For example, with the value <code>http://www.deltawing.com.au</code> , the DUA would display: <code>www.deltawing.com.au</code> |
| <code>isHTML</code> | Specifies that the attribute's value is a HTML text. For example, with the following value: <code>Press here to access www.deltawing.com.au</code> the DUA would display: <code>Press here to access www.deltawing.com.au</code> NOTE: Using this flag leaves Access Presence vulnerable to cross-site scripting attacks. |

| | |
|---------------------------|--|
| <code>isBinary</code> | Specifies that the attribute's value is a binary and that Access Presence should not display it, but allow it to be downloaded. |
| <code>sortDNValues</code> | For an attribute that has multiple DN values, this flag specifies that Access Presence should sort the values alphabetically on the Expanded Entry page. |

For example:

```
flags {isEmailAddress}
```

delimiters

`delimiters` specifies the set of delimiter characters for the attribute type. It specifies the valid characters that can be used to separate values of the attribute type within a single string. It defaults to a semicolon, and usually does not need to be specified.

Of special interest is the delimiter `cr` (which must be specified using the compound string format in Stream DUA). This delimiter is the best choice for attributes which may contain semicolons, such as `orAddress`. Also, if the first delimiter character is `cr`, the DUA displays the attribute with each value on a new line.

Examples:

```
delimiters {";", cr}
```

This specifies that values of the attribute entered by the user must be separated by either semicolons or new-lines; and that when multiple values are displayed, they are separated by new-lines.

preprocessing

`preprocessing` specifies a list of preprocessing functions for an attribute type. The DUA applies the preprocessing functions whenever the user modifies a value of the attribute.

It only applies to attributes with a string syntax, and is ignored for all others. Structured attributes (such as `orAddress` or `telexNumber`) are represented as strings in the DUAs, but are stored in the DSA in a structured form. Conversion to and from the structured form may result in changes to the value entered by the user, but this is not preprocessing.

The component is a sequence of preprocessing function arguments, each argument is labelled by the function name. Two names are defined for each function, one beginning with 'm' for mandatory, the other with 'op' for optional. Optional functions are ignored if the user has turned off preprocessing. For example:

```
preprocessing {mCompress : NULL, opUniqueValues : NULL}
```

Note that the ordering of the list is significant as it is the order in which the preprocessing functions are executed. Executing a list of functions in one order may not have the same effect as executing them in another.

The functions are described later in this chapter (see *Preprocessing functions* on page 109).

attrib-hiding

Deprecated.

replace-strings

Deprecated.

description

Describes how the attribute should be used and is displayed on Modify page.

dateTimeFormat

Defines the date/time presentation of an attribute that has a GeneralizedTime or UTCTime syntax.

The format string can include the following specifiers (plus arbitrary text):

- ❑ %% – replaced with a literal '%' character
- ❑ %C – century number [0,99], two characters padded by zero
- ❑ %d – day of month [1,31], two characters padded by zero
- ❑ %D – %m/%d/%y
- ❑ %F – %Y-%m-%d
- ❑ %H – hour (24-hour clock) [0,23], two characters padded by zero
- ❑ %j – day number of year [1,366], three characters padded by zero
- ❑ %m – month [1,12], two characters padded by 0
- ❑ %M – minute [0,59], two characters padded by 0
- ❑ %R – %H:%M
- ❑ %s – fraction of second
- ❑ %S – seconds [0,60], two characters padded by zero
- ❑ %T – %H:%M:%S
- ❑ %y – year without century [0,99], two characters padded by zero
- ❑ %Y – year with century
- ❑ %z – time zone as hour/minute offset from UTC, four characters padded by zero preceded by '+' or '-' sign or 'Z' for UTC
- ❑ %Z – time zone as hour/minute offset from UTC in XML format, '+' or '-' followed by 2 digits for hour, a colon ':' and the 2 digits for minutes

Examples

The following Stream DUA script reads all values of `attributePresentation`:

```
read {
    organizationName "Deltawing" }
return { attributePresentation };
```

The following script adds an `attributePresentation` value for the attribute `surname`:

```
modify {
    organizationName "Deltawing"
    / commonName "Subschema"
}
with changes {
    add values attributePresentation {
```

```

        type surname,
        display-names {{name "Surname"}},
        flags {},
        preprocessing {opCompress},
        attrib-hiding {},
        replace-strings {}
    }
};

```

fields

Controls the presentation of the component fields of a complex attribute value. This information is used primarily on the Modify Value Form, although some fields also affect the Expanded Entry page.

| | |
|--------------------------|--|
| <code>name</code> | The name of the field in the syntax definition. |
| <code>label</code> | The display name to use for the field on the Modify Value Form. |
| <code>behaviour</code> | Provides information about how field values should be presented, as described below: |
| <code>normal</code> | The default presentation for the syntax - a text input control. |
| <code>formatted</code> | A textarea input control. |
| <code>constrained</code> | Allows a list of permitted values for the field to be provided. These will be presented in a select input control. |
| <code>uuid</code> | Indicates that the field holds a value that should be treated as a string representation of the entryUUID attribute value of an entry in the directory. Access Presence will display such fields in the way it does for a DistinguishedName, using the entry cache to provide candidate values on the Modify Value Form. |
| <code>uri</code> | Indicates that the field holds a URI that should be presented as a link on the Expanded Entry page. |

objectClassPresentation

This multi-valued operational attribute specifies how directory entries are displayed in the Expanded Entry page of Access Presence.

It has the following ASN.1 definition:

```
objectClassPresentation ATTRIBUTE ::= {
    WITH SYNTAX          ObjectClassPresentation
    EQUALITY MATCHING RULE objectIdentifierFirstComponentMatch
    USAGE                 directoryOperation
    ID                    {vf 18 6} }

ObjectClassPresentation ::= SEQUENCE {
    object-class          [0]      OBJECT-CLASS.&id,
    displayName           [1]      TeletexString,
    sub-classes           [2]      SEQUENCE OF Subord OPTIONAL,
    expanded-atts         [3]      SEQUENCE OF ATTRIBUTE.&id,
    special-att           [4]      AttNumb OPTIONAL,
                                -- number is column position --
    modify-atts           [5]      SEQUENCE OF ATTRIBUTE.&id,
    inherit-atts          [6]      SEQUENCE OF ATTRIBUTE.&id OPTIONAL,
    username-atts         [7]      SEQUENCE OF AttNumb OPTIONAL,
                                -- number is #chars to use (0 is all) --
    exp-name-label        [8]      TeletexString OPTIONAL,
    rdn-atts              [24]     SEQUENCE OF ATTRIBUTE.&id OPTIONAL,
    mand-atts             [25]     SEQUENCE OF ATTRIBUTE.&id OPTIONAL,
    opt-atts              [26]     SEQUENCE OF ATTRIBUTE.&id OPTIONAL,
    useNamingOptionals    [9]      BOOLEAN DEFAULT FALSE,
    reverseLink           [10]     SEQUENCE OF ReverseLink OPTIONAL,
    relatedEntries        [11]     RelatedEntries OPTIONAL,
    preferredName         [12]     ATTRIBUTE.&id OPTIONAL,
    request-atts          [13]     SEQUENCE OF ATTRIBUTE.&id OPTIONAL
}

ReverseLink ::= SEQUENCE {
    forwardType           ATTRIBUTE.&id,
    reverse               SEQUENCE OF ReverseLinkItem
}

ReverseLinkItem ::= %M:SetReverseLinkItem SEQUENCE {
    type                  ATTRIBUTE.&id,
    objectClass           OBJECT-CLASS.&id OPTIONAL,
    exclude               ReverseLinkOperation OPTIONAL,
    deleteValuesReferencingMoved BOOLEAN DEFAULT FALSE
}

ReverseLinkOperation ::= BIT STRING {
    add      (0),
    remove   (1),
    modify    (2),
    rename    (3),
    move      (4)
}

Subord ::= SEQUENCE {
```

```

-- subClasses has replaced subClass, which exists for
-- compatability with old versions
-- If both are present, subClass is ignored
subClass          OBJECT-CLASS.&id OPTIONAL,
subClasses        SEQUENCE OF OBJECT-CLASS.&id OPTIONAL,
subClassName      TeletexString,
attributeInfo     SEQUENCE OF AttNumb
-- number is width as percentage -- }
AttNumb ::= SEQUENCE {
    type          ATTRIBUTE.&id,
    number        INTEGER }
RelatedEntries ::= SEQUENCE {
    alternativeClass [0] OBJECT-CLASS.&id,
    linkAttribute   [1] ATTRIBUTE.&id,
    searchForm      [2] TeletexString
}

```

Components

object-class

`object-class` specifies the object class to which the value of `objectClass-Presentation` applies. It is either a symbolic name or object identifier.

For example:

```

object-class organizationalUnit
object-class {1 3 32 0 2 4 6 0}

```

displayName

`displayName` specifies a single display name for the object class. It is the name displayed to the user instead of the object class's actual name. For example, the display name for the `organizationalPerson` class might be 'Staff member'.

The display name is used by the DUA when a user adds a new subordinate entry, or moves or deletes an existing entry. As with an attribute, the display name for an object class can be appended with a '%' character followed by an integer to associate it with an icon (see *display-names* on page 89).

For example: `display-name "Unit"`

sub-classes

`sub-classes` specifies information about groups of classes subordinate to the object class. The subordinates within each group are sorted and displayed together.

The information about each group of subordinate classes consists of object identifiers, display names when displayed as subordinates, and the attribute types and their position for display.

NOTE: The number attached to the attribute type is its display width (as a percentage of the total available display width). A width of zero can be specified for an attribute type so that the attribute is not displayed, but is used when sorting results.

The subordinate classes listed must be consistent with the permitted subordinate classes defined by the structure rules in the schema attribute `dITStructureRules` (see the *Technical Reference Guide: Directory System Agent*). However, a

subordinate class specified in `dITStructureRules` can be omitted from this component.

For example:

```
sub-classes {
  {
    subClasses organizationalPerson, organizationalRole,
    subClassName "Staff & Roles",
    attributeInfo {
      {type commonName, number 45},
      {type role, number 33},
      {type telephoneNumber, number 25}
    }
  }
}
```

This example specifies two subordinate classes for the object class whose `objectClassPresentation` is being defined (assumed to be `organizationalUnit`): `organizationalPerson` and `organizationalRole`. The attributes displayed are common to both classes.

expanded-atts

`expanded-atts` specifies the attributes displayed on the Expanded Entry page for entries of the object class. The order in which the attributes are listed determines the order in which they are displayed by the DUA.

For example: `expanded-atts {telephoneNo, location}`

special-att

`special-att` specifies an attribute, such as `telephoneNumber`, to be displayed at the top of the Expanded Entry page and to the right of the entry name.

The column number has been deprecated.

For example: `special-att {type telephoneNumber, number 45}`

modify-atts

`modify-atts` specifies attribute types to be presented for users to modify when adding or modifying an entry. The order in which the attributes are listed determines the order in which they are displayed by the DUA.

This component should be consistent with the mandatory and optional attributes specified through the schema attributes `objectClasses` and `dITContentRules` (see the *Technical Reference Guide: Directory System Agent*). The DUA appends omitted mandatory attributes, and omits specified attributes that are neither mandatory nor optional.

For example:

```
modify-atts {
  organizationalUnitName,
  location,
  telephoneNumber,
  keylink
}
```

inherit-atts

`inherit-atts` specifies attributes to be inherited from an entry's immediate superior when an entry of this object class is created (added).

For example: `inherit-atts {location}`

username-atts

`username-atts` specifies the attribute types and numbers used to construct the user name for entries of this object class. The numbers define the number of characters to use from a value of the specified attribute type (a value of 0 means use the entire attribute value).

The attribute types specified must be mandatory for the object class. That is, they must be specified as mandatory in the `objectClasses` (or `dITContentRules`) schema attributes (see the *Technical Reference Guide: Directory System Agent*).

For example:

```
username-atts {
    {type givenName, number 1},
    {type surname, number 6}
}
```

This example builds user names by concatenating the first character of the attribute `givenName` with the first six characters of the attribute `surname`.

exp-name-label

`exp-name-label` specifies a label for entries of this object class. It is shown next to the entry name in the Expanded Entry page.

For example: `exp-name-label "NAME: "`

useNamingOptionals

`useNamingOptionals` specifies a Boolean flag to indicate whether optional naming attributes should be used to form an entry's DN when values of the optional naming attributes are provided on the Modify page. (Optional naming attributes are identified in the name form for the structural object class of the entry.)

For example: `useNamingOptionals TRUE`

reverseLink

`reverseLink` specifies rules for automatically maintaining DN references between entries. When an entry is referenced by a second entry using a DN value, `reverseLinks` can be used to automatically include a reverse reference from the first entry to the second. The `reverseLinks` rules define when this should happen and what attributes should be used for the reverse link.

The `forwardType` field identifies which attribute type (in the object class the `objectClassPresentation` applies to) should trigger a reverse link in the referenced entry.

The `reverse` field is a list of rules identifying the attribute type (in the referenced entry) to which the DN of the referencing entry should be added. An element in the list with an `objectClass` field will identify the behaviour to use when the referenced entry is of the identified object class. An element in the list without an `objectClass` field identifies the default behaviour.

The `exclude` field identifies operation types for which the reverse-link value should not be maintained. The `deleteValuesReferencingMoved` field, when set to `TRUE`, indicates that if the referenced entry is moved to a new superior, both the forward and reverse links should be removed. For example:

```
reverseLink {
  {
    forwardType roleOccupant,
    reverse {
      type seeAlso,
      deleteValuesReferencingMoved TRUE
    }
  }
}
```

This example defines that when a `roleOccupant` of an `organizationalRole` entry refers to another entry, to identify the occupant of the role, the referenced entry will have a `seeAlso` attribute value added to it identifying the `organizationalRole` referencing it. If the entry referenced by the `roleOccupant` value is moved to another superior, the `roleOccupant` value and the `seeAlso` value referencing the `organizationalRole` entry will be removed.

NOTE: The reverse links and DN tracking functionality overlap considerably (see *dnTracking* in the *Indexes, extensions and word lists* chapter of the *Technical Reference Guide: Directory System Agent*). If configured inconsistently they can behave unpredictably.

RelatedEntries

`RelatedEntries` works in conjunction with `reverseLink` to provide a workflow for Access Presence users when they add multiple entries for the same person. The workflow encourages the user to create roles for a person who already has an entry.

To illustrate, related entries ensure that a person with multiple entries has one main entry of a particular class (`organizationalPerson`, for example) and several related entries of an alternative class (`organizationalRole`, for example). An example is shown in Figure 4 on page 100.

When a user adds a new entry for the above `organizationalPerson`, the workflow is as follows.

1. Access Presence presents the user with a Search Form (the type of Search Form is identified by the `searchForm` field of `RelatedEntries`).
2. The user searches for 'John Self'.
3. The Search Results page lists all entries that fit the search criteria, and displays a button next to each that allows the user to create a new `organizationalRole`. The page also gives the user the option to create a new `organizationalPerson` entry.
4. The user creates a new `organizationalRole` for an entry. Access Presence automatically populates the new entry's attributes with the user's search criteria (so they don't have to enter the same information twice).
5. Access Presence adds a related-entry link to the new `organizationalRole` entry, to all existing `organizationalRole` entries, and to the `organizationalPerson` entry.

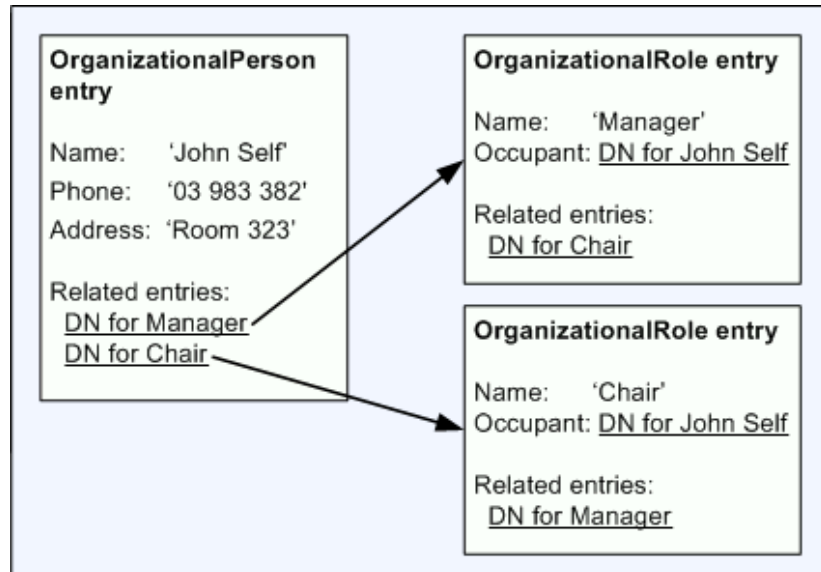


Figure 4: Related entries

`RelatedEntries` has the following fields:

- `alternativeClass` – the object identifier of the object class to use for the subsequent entries for an entity (for example, `OrganizationalRole`).
- `linkAttribute` – when present, identifies an attribute with `DistinguishedName` syntax in the object class identified by the `alternativeClass` field. This enables the link back to the original entry – for example, `roleOccupant` referring back to the `OrganizationalPerson` entry.
- `searchForm` – the name of the Search Form used to determine whether an entity already exists (this should match a `forms.type` field in the `searchOptions` attribute). The Search Form should restrict matching object classes, using the `filter-for` field to specify a single-occurrence object class (for example, `OrganizationalPerson`).

preferredName

A *preferred name* attribute can be defined for a class of entry. The value of the attribute is then used as an entry's label in the Access Presence pages, providing an alternative to using the default label. The default is the entry's mandatory naming attribute defined by schema.

The value of a preferred name attribute also appears in the labels generated by the `VFLabel` tag (see page 47) and for links to Distinguished Names. The preferred name can be accessed through either the Stream DUA or through the ViewDS Management Agent – see the help topic *View or modify an object class's DUA presentation*.

request-atts

This field identifies the attributes that can be accessed by a 'requestor' in the approval process (see page 151).

Examples

The following script reads all values of the `objectClassPresentation` attribute:

```
read { organizationName "Deltawing" }
return { objectClassPresentation };
```

The following Stream DUA script adds an `objectClassPresentation` value for the object class `organizationalUnit`:

```
modify {
    organizationName "Deltawing"
    / commonName "Subschema"
}
with changes {
    add values objectClassPresentation {
        class organizationalUnit,
        displayName "Unit",
        sub-classes {
            {
                subClass organizationalPerson,
                subClassName "Staff",
                attributeInfo {
                    {type commonName, number 45},
                    {type title, number 30},
                    {type telephoneNumber, number 25}
                }
            }
            {
                subClass organizationalUnit,
                subClassName "Unit",
                attributeInfo {}
            }
        },
        expanded-atts { manager, postalAddress, function },
        modify-atts { manager, postalAddress, chargeCode,
            function }
    }
};
```

This sets the display name for the object class to `Unit` and defines how the two subclasses `organizationalPerson` and `organizationalUnit` should be displayed. It declares three attributes to be displayed when the entry is expanded, and four attributes to be displayed for modification.

searchOptions

This single-valued operational attribute specifies the characteristics of the Search Forms, along with other miscellaneous details.

```
searchOptions ATTRIBUTE ::= {
    WITH SYNTAX          SearchOptions
    SINGLE VALUE         TRUE
    USAGE                directoryOperation
    ID                   {vf 18 7} }
SearchOptions ::= SEQUENCE {
    forms                [0] SEQUENCE OF SearchForm,
    context-atts         [1] SEQUENCE OF ATTRIBUTE.&id,
    abbrev-atts          [2] SEQUENCE OF AttribPair OPTIONAL,
                        -- first is abbrev of second --
    flags                [3] BIT STRING {
                        autoExpand(0),
                        topDownSuperiors(1),
                        complexSearching(2) } OPTIONAL,
    pseudo-rdn-level     [4] INTEGER DEFAULT 1,
```

```

        max-att-label-len [5] INTEGER DEFAULT 11 }
SearchForm ::= SEQUENCE {
    type                [0] UnboundedDirectoryString,
    filter-for          [1] SEQUENCE OF OBJECT-CLASS.&id OPTIONAL,
    base-object         [2] DistinguishedName OPTIONAL,
    row1-atts           [3] SEQUENCE OF AttField,
    row2-atts           [4] SEQUENCE OF AttField OPTIONAL,
    row3-atts           [5] SEQUENCE OF AttField OPTIONAL,
    srch-sort-atts      [6] SEQUENCE OF SortKey,
    hierarchyNameBehaviour [7] HierarchyNameBehaviour DEFAULT
    abbreviated,
    orderingStrategy     [8] OrderingStrategy OPTIONAL,
    orderEmptyFieldsAsGreater [9] BOOLEAN DEFAULT FALSE
}
OrderingStrategy ::= ENUMERATED {
    sortKeys (0),
    searchFields (1),
    firstValueExactMatch (2),
    anyValueExactMatch (3)
}
AttField ::= %M:SetAttField SEQUENCE {
    type                ATTRIBUTE.&id,
    number              INTEGER,
    dnAtt               BOOLEAN DEFAULT TRUE,
    component            ComponentField OPTIONAL,
    name                UTF8String OPTIONAL,
    location             [0] AttFieldLocation Default both
}
ComponentField ::= SEQUENCE {
    identifier           ComponentKind,
    rule                MATCHING-RULE.&id
}
ComponentKind ::= CHOICE {
    reference            [0] ComponentReference,
    path                [1] ComponentPath
}
HierarchyNameBehaviour ::= ENUMERATED {
    none                (0),
    abbreviated         (1),
    unabbreviated       (2)
}
AttFieldLocation ::= ENUMERATED {
    request             (1),
    result              (2),
    both                (3)
}

```


Components

SearchOptions

| | |
|---|---|
| <code>forms</code> | Specifies the characteristics for a DUA Search Form. |
| <code>context-atts</code> | <p>Specifies the attribute types that can hold a <i>context</i>. A context is a set of default values entered into the Search Form whenever the user clears the form 'to the context' – for example, a telephone-number prefix.</p> <p>The component is a sequence of attribute types. For example:</p> <pre>context-atts { organizationalUnitName, location, telephoneNumber }</pre> |
| <code>abbrev-atts</code> | <p>Specifies abbreviated attributes to be used in search requests. It is a sequence of <i>attribute pairs</i>: the first attribute in a pair is the abbreviated equivalent of the second.</p> <p>If an attribute type is specified for a Search Form and is also specified by this component as having an abbreviated equivalent, the abbreviated equivalent attribute type (and value) is requested in a search instead of the full attribute type.</p> <p>The abbreviated attributes can be generated by the DSA (for example <code>hiearchyName</code>) but can be any alternative attribute. For instance, you might choose to display a user's initials in place of their given name to save space on the Search Results page.</p> |
| <code>flags</code> <code>pseudo-rdn-</code> <code>level</code> <code>max-att-</code> <code>label-len</code> | Deprecated. |

SearchForm

| | |
|--------------------------|--|
| <code>type</code> | The name of the form – typical values are <code>Name search</code> , <code>Function search</code> , and <code>Unit search</code> . It allows arbitrary search forms, including ones with base objects in other DSAs and other schema administrative areas. |
| <code>filter-for</code> | A list of object classes to include in the search filter constructed from the form. For example, for the <code>Unit search</code> form, it should list the object classes considered to be units. |
| <code>base-object</code> | The base object for the search. If absent, the base object is the same as the configuration-file parameter <code>base-entry</code> . |
| <code>row1-atts</code> | The list of attributes, their column positions (as a percentage of total available width) running across the first row of search fields on the form, and a <code>dnAtt</code> Boolean. Only <code>row1-atts</code> are displayed in the Search Results page (although Access Presence allows the user alter this). |

The `dnAtt` Boolean indicates whether the attributes in the DN of entries should be searched as well as the attributes in the entries themselves. If it is set to `false` then DN will not be searched. This is appropriate if the structure rules allow the attribute type to be present in the DN of entries belonging to the object classes to be searched and such occurrences are to be excluded from the search evaluation.

If multiple attributes are given at the same column in the same row, the attribute is overloaded. This means:

- There is one input field for the set of overloaded attributes.
- A search will try to match values for all the overloaded attributes. For example, if attributes `telephoneNumber` and `extensionNumber` are overloaded, and the user enters a value 3456 on the search form, the DUA will construct a filter `telephoneNumber=3456 or extensionNumber=3456`.
- All the overloaded attributes are requested to be returned from a DSA search.
- The attribute type (out of the set of overloaded attribute types) to be displayed in the search results is the first one for which a value has been returned when scanning through the list of overloaded attribute types in a particular order. This is the order specified by the `disp-pref` component of the `userConfig` attribute in the user's entry (if any) followed by the order of occurrence of the overloaded attributes in the Search Form Attributes.

`row2-atts` The list of attributes and their column positions for the second row of search fields on the form.

`row3-atts` The list of attributes and their column positions for the third row of search fields on the form.

`srch-sort-atts` A list of the attribute types that defines the sort order for search results.

The search results are first sorted by object class to ensure that all entries of the same object class are displayed together. Next, entries of the same object class are sorted on the first attribute type in `srch-sort-atts`. If there are values of the first attribute type that are the same, then values of the second attribute type in `srch-sort-atts` are compared, and so on.

Note that when comparing attribute values, all the values that exactly match the value entered into the corresponding input field are sorted ahead of all other values. The values that do not match the value entered into the input field are sorted lexicographically, and follow the exact matches.

For example:

```
forms {
  {
    type "Name search",
    filter-for {},
    row1-atts {
```

```

        {type surname, number 0},
        {type givenName, number 20},
        {type organizationalUnitName, number 40},
        {type location, number 60},
        {type telephoneNumber, number 80},
        {type extensionNumber, number 80}
    },
    srch-sort-atts { surname, givenName }
}

```

hierarchyName
Behaviour

This field is used to control how `organizationalUnitName` values are displayed on the Search Results page. By default, the `organizationalUnitName` attribute is displayed as a pair of abbreviated values derived from the DN of the displayed entry.

The `hierarchyNameBehaviour` field can be used to alter this default behaviour. When set to `none`, the `organizationalUnitName` behaves like any other attribute: the only `organizationalUnitName` attribute values displayed are those actually defined in the current entry (that is, the DN is ignored). When set to `unabbreviated`, the `organizationalUnitName` is displayed using the same values from the DN as are used in the default behaviour except these values will not be abbreviated.

For example: `hierarchyNameBehaviour none`

ordering
Strategy

This field identifies the search ordering strategy for the Search Results page. (To set the results ordering for a Search Form through the ViewDS Management Agent, see the help topic *View or modify a Search Form*.)

orderEmpty
FieldsAsGreater

Whether Access Presence sorts empty values to the bottom (TRUE) or to the top (FALSE) of the Search Results page.

AttField

The `AttField` type is referenced by the `row1-atts`, `row2-atts` and `row3-atts` fields in the `SearchForm` type. It allows components of complex syntaxes to be identified as separate fields that can be searched individually from a Search Form page. (For information about complex syntaxes, see `attributeSyntax` in the *Schema* chapter of the *Technical Reference Guide: Directory System Agent*.) The referenced component should be a syntax supported for searching – that is, a string, time or integer type.

The `name` in `AttField` allows the name of a search field to be specified. This name is the label displayed to the user, and overrides the attribute display name.

The `location` in `AttField` defines whether a search field is displayed on the Search Form page, Search Results page, or both (the default).

Examples

The following Stream DUA script reads the `searchOptions` attribute:

```
read {
```

```

        organizationName "Deltawing"
    }
    return { searchOptions };

```

The following script modifies an existing value of `searchOptions` by removing the old value and adding a new value:

```

modify {
    organizationName "Deltawing"
    / commonName "Subschema"
}

with changes {
    remove attribute searchOptions,
    add values searchOptions {
        forms {
            {
                type "Name search",
                filter-for {},
                row1-atts {
                    {type surname, number 0},
                    {type givenName, number 20},
                    {type organizationalUnitName, number 40},
                    {type location, number 60},
                    {type telephoneNumber, number 80},
                    {type extensionNumber, number 80}
                },
                srch-sort-atts { surname, givenName }
            }
        },
        context-atts {
            organizationalUnitName,
            location,
            telephoneNumber
        },
        flags {autoExpand, topDownSuperiors}
    }
};

```

This example defines a single Search Form called 'Name search' and sets up five search fields for the attributes `surname`, `givenName`, `organizationalUnitName`, `location`, and `telephoneNumber`. The last of these attributes is overloaded with `extensionNumber`. Results will be sorted by `surname`, then `givenName`. The third, fourth and fifth of these attributes are set up as context attributes.

defaultEntitlement

This single-valued operational attribute specifies limits and functional capabilities for a user. Access Presence enforces these limits (however, other DUAs may not observe them).

This attribute should be placed in the *subschema subentry*. It provides a system-wide limit for all users. However, it can be overridden for a specific user by placing the `userEntitlement` attribute (which has the same syntax) in the user's entry.

This attribute has the following ASN.1 definition:

```

defaultEntitlement ATTRIBUTE ::= {
    WITH SYNTAX UserEntitlement

```

```

SINGLE VALUE      TRUE
USAGE            directoryOperation
ID              {vf 18 12} }
UserEntitlement ::= SEQUENCE {
    limit          [1] Limit OPTIONAL,
    func-cap       [2] FunctionalCapabilities OPTIONAL,
    gc-limit       [3] Limit OPTIONAL
    -- for global changes: timeout is actually
    -- subsearch limit -- }
Limit ::= SEQUENCE {
    time-limit     [0] INTEGER,
    size-limit     [1] INTEGER,
    timeout        [2] INTEGER }
FunctionalCapabilities ::= BIT STRING {
    globalChanges(0),
    sortSubs(1),
    moveMultipleSubs(2),
    viewUsers(3),
    printing(4),
    statistics(5),
    billing(6),
    viewLogs(7),
    allowSavePassword(8) }

```

Components

limit

limit specifies several time and size limits, and is a sequence of three integers:

- *time-limit* is the time limit in seconds to request for each DSA operation. A typical value is 30 seconds.
- *size-limit* is the size limit (number of entries to return) to request for a search-one-level operation (i.e. listing immediate subordinates).
- *timeout* is the maximum inactive time in seconds, after which the DUA will disconnect from the DSA. A typical value is 1800 seconds (30 minutes).

If this component is absent (and is not supplied by a value of *userEntitlement* in the user's entry), the DUA does not set any limits never times out.

For example:

```
limit { time-limit 30, size-limit 99, timeout 1800 }
```

func-cap

func-cap specifies functional capabilities available to a user with update or administrator access. (Super-users have implicit access to all functional capabilities provided by the DUA.)

The following functional capabilities can be specified:

| Identifier | Description |
|---------------|--|
| globalChanges | Allows the user to modify all entries within their assigned subtree that satisfy certain selection criteria with a single command. |

| Identifier | Description |
|--|---|
| sortSubs | Allows the user to change the sort order of subordinates on an expanded entry form (for example, by selecting them and moving them to the top of the list of entries for that subordinate class). |
| moveMultipleSubs | Allows the user to select multiple subordinate entries on an expanded entry form and then move them to another entry. |
| viewUsers | Deprecated. |
| printing | Allows the user to invoke the Printed Reports command of Access Presence. |
| statistics billing viewLogs allowSavePassword | Deprecated. |

For example:

```
func-cap {globalChanges, sortSubs, printing}
```

gc-limit (not supported by Access Presence)

`gc-limit` specifies time and size limits for the global-changes facility. The global-changes facility allows a user to apply a single command to modify all entries in a specific subtree that satisfy their selection criteria.

To do this efficiently, and to handle when the number of entries exceeds the system limits (set by the parameters `sizelimit` and `timelimit`), the DUA uses a special method of searching for the entries to be modified. It attempts a full subtree search on the base entry. If this search results in a size limit or time limit problem, the search is broken into a listing of immediate subordinates followed by a full subtree search on each subordinate. If these subordinate searches fail with a size limit or time limit, they are handled in the same way, recursively.

For best performance the size limit and time limit requested by the DSA for the full subtree searches may need to be greater than the normal limits for searching. The `gc-limit` component allows suitably relaxed limits to be specified. It is a sequence of three integers:

- `time-limit` is the time limit in seconds to use for global changes searches (typically, 75 seconds).
- `size-limit` is the size limit to use for global changes searches (typically, 500).
- `timeout` is not a timeout at all, but is the size limit to use for global changes subordinate searches (typically, 10000).

If this component is absent (and is not supplied by a value of `userEntitlement` in the user's entry), the DUA does not set any limits in its global changes operation requests, and the DSA's `sizelimit` and `timelimit` apply. The DSA's limits also apply if the specified limits exceed the DSA's.

For example:

```
gc-limit { time-limit 75, size-limit 500, timeout 10000 }
```

Examples

The following Stream DUA script reads the `defaultEntitlement` attribute:

```
read { organizationName "Deltawing" }
return { defaultEntitlement };
```

The following script modifies an existing value of `defaultEntitlement` by removing the old value and adding a new value:

```
modify {
    organizationName "Deltawing"
    / commonName "Subschema"
}
with changes {
    remove attribute defaultEntitlement,
    add defaultEntitlement {
        limit { time-limit 30, size-limit 100, timeout 1800 },
        func-cap {globalChanges, sortSubs, printing},
        gc-limit { time-limit 75, size-limit 500, timeout 10000 }};
```

This sets the system-wide parameters and functional capabilities to the values specified.

Preprocessing functions

The preprocessing functions can only be applied to attributes that have a string syntax (for example, `TeletexString`, `DirectoryString`). A preprocessing function is ignored if it is not specified according to the descriptions in this subsection.

A preprocessing function is described by its ASN.1 tag (which translates to a choice of value) and its value (which supplies an argument to the function).

```
PrepFunction ::= CHOICE {
    opCompress          [0]  NULL,
    mCompress           [1]  NULL,
    opCompare           [2]  TeletexString,
    mCompare            [3]  TeletexString,
    opRemove            [4]  TeletexString,
    mRemove             [5]  TeletexString,
    opReplace           [6]  TeletexString,
    mReplace            [7]  TeletexString,
    opCase              [8]  TeletexString,
    mCase               [9]  TeletexString,
    opReplaceWords      [10] ReplaceStrings,
    mReplaceWords       [11] ReplaceStrings,
    opCompareWords      [12] ReplaceStrings,
    mCompareWords       [13] ReplaceStrings,
    opLength            [14] TeletexString,
    mLength             [15] TeletexString,
    opTruncate          [16] INTEGER,
    mTruncate           [17] INTEGER,
    opUniqueValues      [22] NULL,
    mUniqueValues       [23] NULL,
    opPhone             [24] TeletexString,
    mPhone              [25] TeletexString,
    opCommonName        [26] AttribPair,
```

```

mCommonName      [27] AttribPair,
opBuildAttribute  [28] AttribBuildSpec,
mBuildAttribute   [29] AttribBuildSpec,
opBuildAbbrev     [30] AttribAbbrevSpec,
mBuildAbbrev      [31] AttribAbbrevSpec,
opRegexpMatch     [32] UTF8String,
mRegexpMatch      [33] UTF8String
AttribPair ::= SEQUENCE {
    attrib1          ATTRIBUTE.&id,
    attrib2          ATTRIBUTE.&id }
AttribBuildSpec ::= SEQUENCE {
    string           TeletexString,
    attributes       SEQUENCE OF ATTRIBUTE.&id }
AttribAbbrevSpec ::= SEQUENCE {
    abbrevAtt        [0] ATTRIBUTE.&id,
    sourceAtt        [1] ATTRIBUTE.&id,
    delimiter        [2] TeletexString OPTIONAL }

```

Two additional preprocessing functions are applied implicitly when attribute information is modified. If the attribute is single valued, the DUA checks whether the user entered multiple values, and reports an error if so. (To disable this, explicitly define the delimiters for the attribute to be an empty string.) If the attribute is multi-valued, the DUA checks that the values are all different and removes duplicates if necessary.

Compress

This preprocessing function removes leading and trailing spaces, multiple spaces between words, and unprintable characters (except for carriage-return characters) from an attribute's value.

For example:

```
opCompress: NULL
```

Compare chars

This preprocessing function compares all characters in an attribute's value against a set of specified characters. It reports an error if any characters in the value are not in the set.

It consists of one of the characters A, N, L, D, or U, which specifies a basic character set. Optionally, this is followed by either an I or E, to include or exclude a subsequent set of characters.

| | |
|-----------------|---|
| A | All characters |
| N | No characters |
| L | Alpha characters only |
| D | Digits only |
| U | Alpha-numeric characters only |
| I <i>string</i> | The attribute's value must include any of the characters in <i>string</i> |
| E <i>string</i> | The attribute's value must not include any of the characters in <i>string</i> |

For example, to report an error if punctuation is entered for a new or changed attribute value, specify: `mCompare: "AE., ; ;"`

Remove chars

This preprocessing function compares all characters in an attribute's value against a set of specified characters. It removes any characters from the value that are not in the set.

The supplied value specifies the set of permitted characters, which is defined in the same way as for the *Compare chars* function.

For example, to remove punctuation from value, specify: `opRemove: "AE., ; ;"`

Replace chars

This preprocessing function replaces certain characters in an attribute's value with other characters.

The supplied value specifies the substitutions. It has the form: `RcharsWchars`

The characters following the `R` specify the characters to be replaced; the characters following the `W` are the replacements. For example, to replace all space characters with underscores and all `#` characters with `$` characters, specify: `opReplace: "R #W_$"`

Case

This preprocessing function converts the characters in an attribute value to upper, lower, or mixed case.

The conversion algorithm for mixed case makes some assumptions about the usage of certain punctuation characters – for example, an apostrophe is assumed to be used as in the example, D'Rosario. There is no special recognition of certain standard naming prefixes such as 'Mac'. Therefore, a name such as 'MacKenzie' would be converted to 'Mackenzie'. These sorts of problems in the mixed case conversions mean that it is preferable to only offer this preprocessing function in an optional application. It can then be overridden by a DUA user.

The supplied value specifies the type of conversion:

- `U` Convert to upper case
- `L` Convert to lower case
- `M` Convert to mixed case (the first letter of each word in upper case; and other letters in lower case)

For example, to convert an attribute value to mixed case: `opCase: "M"`

Replace words

This preprocessing function replaces certain words in an attribute value. Words that appear in the replacement list are replaced by a replacement word.

For example, to convert addresses to a standard abbreviated form:

```
opReplaceWords: {
    { replace "Road", with "Rd." },
    { replace "Street", with "St." },
    { replace "Avenue", with "Ave." }
}
```

Compare words

This preprocessing function compares the words in an attribute value with a replacement list. It reports an error if there is a word in the attribute value that is not in the list.

The `with` words should be empty strings.

For example, to restrict values of an attribute to either Dr, Mr, Mrs, Ms, or Miss:

```
opCompareWords: {
    { replace "Dr", with "" },
    { replace "Mr", with "" },
    { replace "Mrs", with "" },
    { replace "Ms", with "" },
    { replace "Miss", with "" }
}
```

Access Presence displays the compare words for an attribute as permitted values in a list box. Additional constraints apply to the attribute if specified in the `attributeSyntax` field of the `attributeType` operational attribute (see `attributeSyntax` in the *Schema* chapter of the *Technical Reference Guide: Directory System Agent*).

Length

This preprocessing function checks whether the length of an attribute value is within specified bounds, and reports an error if it is not.

The supplied value specifies the bounds:

| | |
|--|---|
| <code>Lnum</code> | <code>length ≤ num</code> |
| <code>Gnum</code> | <code>length ≥ num</code> |
| <code>GZnum</code> | <code>length ≥ num</code> or <code>length = 0</code> |
| <code>Lnum1Gnum2</code> or <code>Gnum2Lnum1</code> | <code>length ≤ num₁</code> and <code>length ≥ num₂</code> |
| <code>Lnum1GZnum2</code> or <code>GZnum2Lnum1</code> | <code>length ≤ num₁</code> and <code>length ≥ num₂</code> or <code>length = 0</code> |

For example, to specify that attribute values are to be either five or six characters long:

```
opLength: "G5L6"
```

To specify an exact length, use `LnumGnum` with both `num` terms set to the exact length.

Truncate

This preprocessing function truncates an attribute value to the length specified by the supplied value (an integer). For example, to specify that an attribute value is to be truncated to eight characters: `opTruncate: 8`

Unique values

This preprocessing function checks that the values in a string are unique within an area of the DIT on the local DSA. If they are not unique, the function reports an error.

NOTE: As applying this function involves a search of the directory, the attribute must be indexed.

For example: `opUniqueValues: NULL`

Phone

This preprocessing function checks whether an attribute value has one of the specified telephone number syntaxes. If the value conforms to one of the syntaxes, the function preprocesses the value to a standard format (also specified by the syntax it matches). If it does not conform, the function reports an error.

Each specified syntax defines an acceptable pattern of numbers. It is a string containing one or more patterns, separated by semi-colons. When more than one pattern is specified, strings are compared for matches from left to right and the first match is selected for preprocessing.

A telephone number syntax can contain the following characters:

| | |
|---------------------------|---|
| <code>d</code> | Pattern matches a digit (0-9) – for example, <code>(dd) ddd dddd</code> |
| <code>a</code> | Pattern matches an alphanumeric character – for example, <code>ddd-aaa</code> |
| <code>0123456789</code> | Any digit may be placed in specific position in the string and an entered value must match exactly – for example, <code>13 dd dd</code> |
| <code>[...]</code> | Pattern matches any single character within <code>[]</code> – for example, <code>041[789] ddd ddd</code> |
| <code>() + - space</code> | Need not be present but would be inserted if not present and removed if not appropriate. |

Note that leading and trailing spaces in the string are inserted into the formatted telephone number.

In the special case where two consecutive format strings are equal (when ignoring the formatting characters `() + - space`) then the positions of any spaces will determine the matching format which will be used for preprocessing. That is, the format string that matches the most consecutive spaces from the beginning will be selected.

For example: `mPhone: "13 dd dd; (dd) ddd dddd; (ddd) ddd ddd"`

With this example:

- 07 2584505 would be formatted to (07) 258 4505;
- 072 584505 would be formatted to (072) 584 505; and
- 072584505 would be formatted to (07) 258 4505 because of the ordering of the format string.

Common name

This preprocessing function checks whether a value contains the values of two other attributes, and reports an error if it does not. It should be specified as one of the preprocessing functions for the `commonName` attribute to ensure that its value contains the `surname` and one of the `givenName` attribute values of the entry.

For example: `opCommonName: { attrib1 givenName, attrib2 surname }`

Build attribute

This preprocessing function builds attribute values from a combination of other attribute values or constant text. Its main purpose is to allow the `commonName` attribute value to be modified automatically whenever one of the `personalTitle`, `givenName`, or `surname` attribute values are modified.

The supplied value consists of a TeletexString and a list of attributes, and specifies how the first attribute in the `attribute` list is built. The string can include the following:

- `%` – the built attribute is formed by replacing each `%` character with the next attribute in the `attributes` list
- `~` – an escape character that allows a literal `%` or `~` to be included in the built attribute
- constant text

For example:

```
opBuildAttribute: {
    string "% % %",
    attributes { commonName, personalTitle, givenName, surname } }
```

Abbreviate attribute

This preprocessing function builds an attribute value by abbreviating other attribute values (it takes the first letter from each). A common use of this function is to automatically update an `initials` attribute when a value of the `givenName` attribute is created or modified.

The supplied value consists of two attributes and a delimiter:

- `abbrevAtt` names the abbreviated attribute to be generated or maintained
- `sourceAtt` names the attribute which supplies the non-abbreviated value
- `delimiter` is typically a `'.'` or `' '`

The `abbrevAtt` attribute is formed by dividing the value of `sourceAtt` into words separated by spaces, taking the first letter of each word, and concatenating them with the delimiter (if present) between each letter and after the last.

This preprocessing function should be specified for `sourceAtt`.

For example:

```
opAbbrevAtt: {
    abbrevAtt  initials,
    sourceAtt  givenName,
    delimiter  '.' };
```

Regular expression match

This preprocessing function verifies an attribute's value according to whether it matches a provided regular expression (that conforms to the XPath specification).

The regular expression is not anchored. The `^` and `$` characters are the start and end anchors respectively.

User operational attributes

Access Presence reads attributes from a user's entry, which affect Access Presence's operation for that user. They are:

- `userEntitlement`
- `userConfig`
- `privilege`

The above attributes are described in this section, except for `privilege` which is described in the *Managing Security* chapter of the *Technical Reference Guide: Directory System Agent*.

userEntitlement

This single-valued operational attribute specifies limits and functional capabilities assigned to a particular user. Access Presence enforces these limits (other DUAs may not).

To apply this attribute's limits to a user, place it in the user's entry. If a user's entry does not have this attribute, the system-wide limit defined by `defaultEntitlement` in the subschema subentry applies. The attribute has the following ASN.1 definition (for the attribute's syntax, see `defaultEntitlement` on page 106):

```
userEntitlement ATTRIBUTE ::= {
    WITH SYNTAX      UserEntitlement
    SINGLE VALUE     TRUE
    USAGE            directoryOperation
    ID               {vf 18 8} }
```

Examples

The following Stream DUA script reads the `userEntitlement` attribute for the entry `organizationName "Deltawing" / commonName "John Smith"`:

```
read {
    organizationName "Deltawing"
    / commonName "John Smith" }
return { userEntitlement };
```

The following script modifies an existing value of `userEntitlement` by removing the old value and adding a new value:

```
modify {
    organizationName "Deltawing"
    / commonName "John Smith" }
with changes {
    add userEntitlement {
        limit { time-limit 30, size-limit 100, timeout 1800 },
        func-cap {globalChanges, sortSubs, printing},
        gc-limit { time-limit 75, size-limit 500, timeout 10000 }
    }
};
```

This sets the user's parameters and functional capabilities to the values specified.

NOTE: The values of `userEntitlement` can also be added to user entries by a user with `admin` or `superuser` privilege using Access Presence.

userConfig

This operational attribute records the user's preferred settings for several user-specific options in Access Presence.

```
userConfig ATTRIBUTE ::= {
    WITH SYNTAX          UserConfig
    SINGLE VALUE          TRUE
    USAGE                  directoryOperation
    ID                     {vf 18 9} }
UserConfig ::= SEQUENCE {
    context                [0] SEQUENCE OF AttString OPTIONAL },
    default-base           [1] DistinguishedName OPTIONAL
    disp-pref              [2] SEQUENCE OF ATTRIBUTE.&id OPTIONAL,
    language               [3] Language OPTIONAL,
    codepage               [4] CodePage OPTIONAL }      DEPRECATED
AttString ::= SEQUENCE {
    attribute              ATTRIBUTE.&id,
    valueString            TeletexString }
```

context

`context` specifies the user's context at DUA startup. It is a list of attribute types with string values. If these attribute types are in a Search Form, and the context is enabled, then the string values are displayed in the appropriate fields so that the user does not have to enter them.

For example: `context {{attribute telephoneNumber, value "(03) 9876"}}`

default-base

`default-base` specifies a base object to use in directory searches in preference to the base-entry specified in the configuration file. If used, it typically specifies a sub-unit below the base entry.

For example:

```
default-base { organizationName "Deltawing" / organizationalUnit "R&D
Labs" }
```

disp-pref (not supported by Access Presence)

`disp-pref` specifies the user's display preferences on a search form when attributes are overloaded and values of more than one of the overloaded attributes are available.

For example, if `telephoneNumber` and `extensionNumber` are overloaded, and a value is available for both, then the DUA must decide which value to show. It chooses the first attribute in the `disp-pref` sequence of attribute types. It must be set using the Stream DUA.

For example: `disp-pref {extensionNumber, telephoneNumber}`

Approval process operation attributes

The operation attributes for the approval process (see page 151) are described below.

```
viewDSUpdateRequest ATTRIBUTE ::= {
    WITH SYNTAX UpdateRequest
    EQUALITY MATCHING RULE directoryStringFirstComponentMatch
    USAGE directoryOperation }
```

```

ID id-viewDS-oa-updateRequest
}
UpdateRequest ::= SEQUENCE {
    identifier [0] UnboundedDirectoryString,
        -- Recommended that this is a UUID in string representation
    activity    [1] SEQUENCE SIZE (1..MAX) OF UpdateActivity,
        -- Every request should include a creation activity record
    details     [2] UpdateRequestDetails,
    object      [3] DistinguishedName OPTIONAL
        -- Object name only required when:
        -- the request is not stored in the entry it applies to;
        -- and it is not an add subordinate request
}
UpdateActivity ::= SEQUENCE {
    action      [0] UpdateAction,
    name        [1] DistinguishedName,
    at          [2] GeneralizedTime,
    description [3] UTF8String OPTIONAL
}
UpdateAction ::= ENUMERATED {
    created      (0),
    updated      (1),
    approved     (2),
    rejected     (3),
    cancelled    (4),
    notification (5)
}
UpdateRequestDetails ::= CHOICE {
    add [0] SEQUENCE {
        structural [1] OBJECT-CLASS.&id,
        nameform   [2] NAME-FORM.&id OPTIONAL,
        content     [3] SET OF Attribute
            -- The distinguished values will be determined from the
            -- entry content and schema when the request is approved.
            -- The parent is determined by the entry the request
            -- is stored in.
    },
    remove [1] NULL,
    move   [2] DistinguishedName,
        -- The distinguished name of the new superior
    modify [3] SET OF Attribute
}

```

The set of attributes for the modify request lists all the attributes modified in the request and the resulting set of values for each attribute. This will include attributes with no values, indicating that the attribute was removed.

New entry operation attributes

These operational attributes relate to when a user creates a new entry:

- newSubordinateModifyRights
- permittedNewSubordinates

- permittedImports

newSubordinateModifyRights

Access Presence uses this attribute when a user adds a new entry. It ensures that the user is presented with only the attributes they are permitted to access according to the access-control scheme.

```
newSubordinateModifyRights ATTRIBUTE ::= {
    WITH SYNTAX NewSubordinateModifyRights
    SINGLE VALUE TRUE
    NO USER MODIFICATION TRUE
    USAGE directoryOperation
    ID id-adacel-oa-newSubordinateModifyRights
}

SubordinateModifyRights ::= %M:SetSubordinateModifyRights SEQUENCE {
    structural OBJECT-CLASS.&id,
    auxiliaries SET OF OBJECT-CLASS.&id,
    modifyRights ModifyRights
}

NewSubordinateModifyRights ::= SEQUENCE OF SubordinateModifyRights
```

A `SubordinateModifyRights` element is created for each combination of auxiliary object class values that are permitted by the schema and the access controls.

When a user attempts to add a new subordinate entry, then the parent entry's `newSubordinateModifyRights` is obtained. This describes the modify rights for a subordinate entry, which allows Access Presence to display an Add page that takes the appropriate access controls into account.

It also allows Access Presence to populate the `objectClass` attribute of a new entry with suggested auxiliary object class values. However, this only occurs if the `newSubordinateModifyRights` indicates that adding the structural object class without values of an auxiliary object class is not permitted.

permittedNewSubordinates

For each entry, this attribute lists the *name forms* for entries that can be created as subordinates to the entry. If the access controls permit no subordinates, an empty list is provided. If the DSA cannot determine a suitable list, the attribute is not returned.

```
PermittedNewSubordinates ::= SEQUENCE OF NAME-FORM.&id

permittedNewSubordinates ATTRIBUTE ::= {
    WITH SYNTAX PermittedNewSubordinates
    SINGLE VALUE TRUE
    NO USER MODIFICATION TRUE
    USAGE directoryOperation
    ID id-adacel-oa-permittedNewSubordinates
}
```

permittedImports

For each entry, this attribute lists the *name forms* for existing entries that the user can move within the DIT to become subordinate to the entry. If the access controls permit no subordinates, an empty list is provided. If the DSA cannot determine a suitable list, the attribute is not returned.


```

PermittedImports ::= SEQUENCE OF NAME-FORM.&id
permittedImports ATTRIBUTE ::= {
    WITH SYNTAX                PermittedImports
    SINGLE VALUE                TRUE
    NO USER MODIFICATION      TRUE
    USAGE                      directoryOperation
    ID                        id-adacel-oa-permittedImports
}

```

Other operational attributes

ViewDS has a number of ViewDS-specific user and operational attributes which are used by Access Presence in special ways:

- `sortSubs`
- `hierarchyName`
- `unabbreviatedHierarchyName`
- `updatersName`
- `viewDSMatchQuality` and `viewDSSimpleMatchQuality`
- `viewDSSessionObject`

Finally, the following operational attributes are not used directly by Access Presence, but are used by the DSA to generate information used by Access Presence:

- `hierarchyNameSpecification`
- `resolvedDistinguishedName`

sortSubs

`sortSubs` is a simple text string used by Access Presence as a sort-key when listing subordinates of a given object class. It allows designated people or units to appear ahead of their natural sort position (typically alphabetical-by-name).

It is a user attribute that must be defined in the schema to appear in entries. The DUA will request the attribute if it is defined in the schema.

It should not normally be included in `attributePresentation`.

```

sortSubs ATTRIBUTE ::= {
    WITH SYNTAX                PrintableString
    EQUALITY MATCHING RULE caseExactMatch
    ORDERING MATCHING RULE caseExactOrderingMatch
    SUBSTRINGS MATCHING RULE caseExactSubstringsMatch
    SINGLE VALUE                TRUE
    ID                        {vf 4 0}
}

```

hierarchyName

This operational attribute is a text string that holds an abbreviated representation of an entry's superior units.

The DSA constructs the value from the entry's DN, and according to the specification in the `hierarchyNameSpecification` attribute that is in the subschema administrative area containing the entry.

The `hierarchyName` attribute is displayed on a Search Form wherever the forms component defines that an `organizationalUnitName` attribute should be displayed. If the entry can have subordinates, the DUA replaces the second component with the RDN of the entry itself. Both RDNs are abbreviated.

```
hierarchyName ATTRIBUTE ::= {
    SUBTYPE OF          name
    WITH SYNTAX          DirectoryString { ub-name }
    SINGLE VALUE         TRUE
    NO USER MODIFICATION TRUE
    ID                   {vf 4 1}
}
```

hierarchyNameSpecification

This operational attribute can be used to control the components of a DN which construct the attributes `hierarchyName` and `unabbreviatedHierarchyName`. The `hierarchyNameSpecification` should be defined in a subschema subentry.

```
hierarchyNameSpecification ATTRIBUTE ::= {
    WITH SYNTAX          HierarchyNameSpecification
    EQUALITY MATCHING RULE objectIdentifierFirstComponentMatch
    USAGE                dSAOperation
    ID                   { ads 18 2 }
}

HierarchyNameSpecification ::= SEQUENCE {
    structuralObjectClass OBJECT-CLASS.&id,
    delimiter              DirectoryString{ub-name}
                           DEFAULT UTF8String:";",
    topBound               [0] HierarchyNameSelection OPTIONAL,
    bottomBound            [1] HierarchyNameSelection OPTIONAL,
    specification          SEQUENCE OF HierarchyNameComponent
}

HierarchyNameComponent ::= SEQUENCE {
    selection              HierarchyNameSelection,
    preferredAVA           SEQUENCE OF ATTRIBUTE.&id OPTIONAL
}

HierarchyNameSelection ::= CHOICE {
    top                   [0] INTEGER,
    autonomous           [1] INTEGER,
    bottom                [2] INTEGER }
}
```

The `hierarchyNameSpecification` is a multi-valued attribute, where each value defines the rules for building `hierarchyName` and `unabbreviatedHierarchyName` for a particular structural object class.

The exception is a value of `hierarchyNameSpecification` defined for the object class `top`. This value of `hierarchyNameSpecification` is the default rule used if a specific rule is not defined for a structural object class. The `structuralObjectClass` field identifies which object class the `hierarchyNameSpecification` value applies to.

The `delimiter` field identifies a character sequence to be used to separate components in the value of `hierarchyName` or `unabbreviatedHierarchyName`. The default is semi-colon character.

The `specification` field is a sequence of structures that define specific RDNs in the DN to include in the constructed attribute. Each RDN is referenced by a positive numeric index relative to one of three reference points.

The reference points are:

- `top` – the first RDN of the DN under the root entry. The offset from this point indexes towards the end of the DN. Thus, the reference `top:0` indicates the first RDN under the root entry.
- `bottom` – the last RDN of the DN. The offset from this point indexes towards the start of the DN. Thus, the reference `bottom:1` indicates the second last RDN in the DN.
- `autonomous` – this reference point is relative to the RDN of the autonomous administrative point which applies to the entry being evaluated. For example, in the demonstration database Deltawing, the O "Deltawing" entry is an autonomous point. The `hierarchyNameSpecification` defined for the O "Deltawing" subschema area uses the O "Deltawing" entry as the point for evaluating the `autonomous` references, regardless of any superior entry in the DIT. The offsets relative to the autonomous point index towards the end of the DN.

The `preferredAVA` field in each `specification` structure is optional. It indicates the preferred attribute type to use in the constructed value when a selected RDN has more than one `AttributeTypeAndValue` in the RDN. The first attribute type in the list which corresponds to a type occurring in the RDN is selected. If none match, or the `preferredAVA` field is not specified, the first attribute in the RDN is selected.

The `topBound` and `bottomBound` provide limits on the RDN components selected. If a component selected by the `specification` field:

- is above the `topBound` or below the `bottomBound`, it is not used in the constructed value.
- matches the `topBound` or `bottomBound`, or is between the two bounds, it is used in the constructed value.

If `hierarchyNameSpecification` is not provided in a subschema administrative area, the default behaviour for constructing `hierarchyName` and `unabbreviated HierarchyName` matches the following `hierarchyNameSpecification` specification.

```
{
    structuralObjectClass top,
    topBound autonomous:0,
    bottomBound bottom:1,
    specification {
        {
            selection autonomous:1
        },
        {
            selection bottom:1
        }
    }
},
{
    structuralObjectClass organizationalUnit,
    topBound autonomous:0,
```

```

        bottomBound bottom:0,
        specification {
            {
                selection autonomous:1
            },
            {
                selection bottom:0,
                preferredAVA {
                    organizationalUnitName
                }
            }
        }
    }
}

```

updatersName

This operational attribute a text string derived from the RDN of the user who most recently modified a directory entry. As it is not automatically maintained by the DUA or DSA, the `modifiersName` attribute should be used in its place.

```

updatersName ATTRIBUTE ::= {
    SUBTYPE OF          name
    WITH SYNTAX          DirectoryString { ub-name }
    SINGLE VALUE         TRUE
    NO USER MODIFICATION TRUE
    USAGE                directoryOperation
    ID                   {vf 18 1}
}

```

viewDSMatchQuality and viewDSSimpleMatchQuality

These attributes allow search results to be sorted according to how well they match a DUA's search filter. They have the following schema definitions:

```

viewDSMatchQuality ATTRIBUTE ::= {
    WITH SYNTAX INTEGER
    EQUALITY MATCHING RULE integerMatch
    ORDERING MATCHING RULE integerOrderingMatch
    SINGLE VALUE TRUE
    USAGE directoryOperation
    ID { 1 3 6 1 4 1 21473 5 18 11 } }

viewDSSimpleMatchQuality ATTRIBUTE ::= {
    WITH SYNTAX INTEGER
    EQUALITY MATCHING RULE integerMatch
    ORDERING MATCHING RULE integerOrderingMatch
    SINGLE VALUE TRUE
    USAGE directoryOperation
    ID { 1 3 6 1 4 1 21473 5 18 12 } }

```

Each entry in a search result has a value for `viewDSMatchQuality` and `viewDSSimpleMatchQuality`. They can be requested by a DUA so that it can sort the results itself; or be referenced by a DUA through a sort control so that the DSA can sort the results.

If an entry matches the search filter exactly, then:

- `viewDSSimpleMatchQuality` equals 0
- `viewDSMatchQuality` equals 0

If an entry matches the search filter approximately, then:

- `viewDSSimpleMatchQuality` equals 1
- `viewDSMatchQuality` is greater than 0 – the higher the value, the less well the entry matches the search filter.

Typically, a DUA issues a sort control that requests sorting on `viewDSSimpleMatchQuality` first and then on the second search attribute (for example, `surname`). This would group the exact matches at the start of the results, followed by the approximate matches in alphabetical order.

viewDSSessionObject

This operational attribute stores session objects in the directory. It is automatically indexed for its equality matching rule and dn-tracking is enabled for it. This attribute should be included in the attributes replicated to another DSA if Access Presence traffic is load-balanced across a master and its replicas. The VMA will include it in the default set of attributes it offers when creating a new replication agreement.

The `viewDSSessionObject` operational attribute has the following schema definition:

```
viewDSSessionObject AttributeTypeDescription ::= {
    identifier { 1 3 6 1 4 1 21473 5 18 20 },
    name { printableString:"viewDSSessionObject" },
    information {
        equalityMatch directoryStringFirstComponentMatch,
        attributeSyntax printableString:"SessionObject",
        multi-valued TRUE,
        application directoryOperation
    }
}

SessionObject ::= SEQUENCE {
    identifier          PrintableString,
    -- Recommend use of a UUID in string representation
    created             INTEGER,
    -- Unix style seconds since 1970-01-01T00:00:00
    subject             DistinguishedName,
    -- Authenticated identity for this session
    remote              UTF8String OPTIONAL
    -- Proxy authorisation remote user identifier
}
```

unabbreviatedHierarchyName

This operational attribute is a text string that holds a representation of an entry's superior units.

The DSA constructs the value from the entry's DN, and according to the specification in the `hierarchyNameSpecification` attribute that is in the subschema administrative area containing the entry.

The `unabbreviatedHierarchyName` attribute is displayed on a Search Form wherever:

- the `forms` component defines that an `organizationalUnitName` attribute should be displayed; and
- the `hierarchyNameBehaviour` component is set to `unabbreviated`.

If the entry can have subordinates, the DUA replaces the second component with the RDN of the entry itself. Both RDNs are abbreviated.

```
unabbreviatedHierarchyName ATTRIBUTE ::= {
    SUBTYPE OF          name
    WITH SYNTAX          DirectoryString { ub-name }
    SINGLE VALUE         TRUE
    NO USER MODIFICATION TRUE
    ID                   {ads 4 0}
}
```

resolvedDistinguishedName

This is an integer corresponding to the internal 'entry identifier' used to tag every entry in the DSA. It is used in constructing an alias name of an entry which is much shorter than the entry's full Distinguished Name. A ViewDS DSA will return such an alias name if the DUA requests the attribute `resolvedDistinguishedName` in a search request, typically to reduce the amount of data transmitted in search results when operating over a slow network.

The attribute is predefined in the schema and is not user modifiable. It does not exist in any entry; it is used only as a signal to the DSA to return such short alias names and in constructing such alias names.

If this attribute is requested, then the DUA should also request the `dontUseCopy` service control for correct operation in the presence of replicated data.

```
resolvedDistinguishedName ATTRIBUTE ::= {
    WITH SYNTAX          ResolvedDistinguishedName
    EQUALITY MATCHING RULE integerMatch
    SINGLE VALUE         TRUE
    NO USER MODIFICATION TRUE
    USAGE                dSAOperation
    ID                   {vf 18 0}
}

ResolvedDistinguishedName ::= INTEGER
```

Chapter 7

Printing DUA

The Printing DUA allows data to be extracted from a directory and formatted into reports, printed directories and formats suitable for other systems. The input to the Printing DUA is an input script – which can be selected by an Access Presence user to generate a report – and its output can be written to a file and displayed by Access Presence.

This chapter has the following sections:

- Running the Printing DUA
- Input script syntax
- Supported attribute syntaxes

Running the Printing DUA

The Printing DUA can be invoked from Access Presence (see *Configuring for printing* on page 25) or from the command line as follows:

```
pdua [ -t vfhome ] [ -a | -u username -p password ]  
[ -b baseobject ] [ -r requestor ] [ -o address ]  
[ -e errorfile ] [ -f outputfile ] [ inputfile ]
```

The command-line options are as follows:

| | |
|----------------------------|--|
| <code>-t vfhome</code> | Sets the ViewDS root directory to <code>vfhome</code> instead of the environment variable <code>\${VFHOME}</code> . |
| <code>-a</code> | Authenticates with the server using anonymous credentials. |
| <code>-u username</code> | Provides a <code>username</code> with which to authenticate with the DSA. The <code>username</code> is either the value of a <code>viewDSUserName</code> attribute or a DN in Stream DUA notation enclosed in curly brackets. (For information about Stream DUA notation, see the <i>Technical Reference Guide: Directory System Agent</i> .) |
| <code>-p password</code> | Provides a <code>password</code> with which to authenticate with the DSA. |
| <code>-b baseobject</code> | Sets the base object to be the starting point in the DIT from which the Printing DUA will generate a report. This option overrides the base-object option (see <i>base</i> on page 127) in the input script. |

| | |
|---------------------------|--|
| <code>-r requestor</code> | Enable proxy authorisation for each request sent to the DSA, using <code>requestor</code> as the identity that should be used to evaluate access controls. |
| <code>-e errorfile</code> | Sets the name of the file to which the Printing DUA writes error messages. |
| <code>-f filename</code> | Sets the name of the file for all normal output from the Printing DUA. When this option is unspecified, the Printing DUA writes output to the command line (stdout). |
| <code>-o address</code> | Connect to <code>address</code> instead of the address declared by the configuration-file parameter <code>dsaAccessPoint</code> or <code>dsaAddress</code> . |
| <code>inputfile</code> | <p>The Printing DUA reads the input <code>file</code> containing a script. The script declares the entries and attributes to be extracted from the DSA, and the sorting parameters and the tags, text, and formatting to be inserted.</p> <p>By convention, the input file has a name ending in the suffix <code>ds</code>; the output is directed to a file with the suffix <code>do</code>; and stderr if redirected goes to a file with suffix <code>er</code>.</p> |

Input script syntax

The input to the Printing DUA is a *script* specifying the content to be produced. The script consists of a list of parameters, some optional, some mandatory, that must be given in a fixed order. Each parameter begins on a new line and consists of a keyword, followed by a colon, a parameter argument, and a semi-colon.

The input language has the same extended Backus-Naur Form notation as the Stream DUA (see the *Technical Reference Guide: Directory System Agent*):

- | separates alternative expressions
- () groups the enclosed expressions
- { } means zero or more repetitions of the enclosed expression
- [] means the enclosed expression is optional

These symbols are underlined if they form part of the syntax itself; the name of a production is shown in *italic* text; and literal text (e.g. a keyword) is shown in the font Courier New.

Script parameters

An input script consists of the following three groups of parameters in the order shown. Any text following the character `#` is a comment.

Entry extraction parameters

```
base: name;
include_base: ( yes | no );
[depth: number;]
[exclude_subtrees: filter;]
```



```
[select_filter: filter;]
extract_type: (matching | subtree);
preserve_hierarchy: ( yes | no );

# Entry ordering parameters
[class_order: objectclass {, objectclass };]
[entry_sort: class_sort { class_sort };]

# Data output parameters
[file_start: string;]
entry_output: objectclassoutput { objectclassoutput };
[file_end: string;]
[data_formats: format { format };]
```

Entry extraction parameters

The entry extraction parameters specify the region of the DIT (a subtree) from where data is to be extracted.

base

The mandatory parameter `base` specifies the name of the base entry of a subtree. It corresponds to the X.500 Search operation parameter `baseObject`.

```
base: name;
```

Where `name` is the full DN of an entry.

For example:

```
base :
    organizationName "Deltawing"
    / organizationalUnitName "Research Laboratories"
    ;
```

include_base

The mandatory parameter `include_base` indicates whether the base entry should be included in the extraction.

```
include_base: ( yes | no );
```

For example:

```
include_base : yes;
```

depth

The optional parameter `depth` specifies the lower limit of the subtree. If it is unspecified there is no depth limit. A value of zero indicates the base entry.

```
depth: number;
```

The limit is specified as a 'distance' from the base of the subtree. The distance between the base entry and an entry subordinate to it is defined as the number of extra RDN terms in the DN of the latter. The subtree consists of the base entry and all of its subordinates which lie within the distance specified (including those which lie at the limit).

For example:

```
depth : 3;
```

exclude_subtrees

The optional parameter `exclude_subtrees` reduces the DIT subtree region defined by `base`, `include_base` and `depth` by specifying subtrees to be excluded.

```
exclude_subtrees: filter;
```

where `filter` is in the form of an X.500 filter or an LDAP string representation of a filter enclosed in double quotes. Any entry matching the filter together with all of its subordinates will be excluded from the DIT region from which the data is to be extracted.

For example:

```
exclude_subtrees:
    not (objectClass = organizationalUnit or objectClass =
        organizationalPerson);
```

The parameters `select_filter`, `extract_type` and `preserve_hierarchy` specify the entries to be extracted and whether the hierarchical relationships between them are to be preserved.

select_filter

The optional parameter `select_filter` specifies a selection filter that helps define the set of entries to be extracted. If it is missing an empty filter is assumed. The parameter corresponds to an X.500 search filter or an LDAP string representation of a filter enclosed in double quotes. Together with `extract_type`, it specifies the criteria by which entries are selected for extraction.

```
select_filter: filter;
```

For example:

```
select_filter : objectClass = organizationalUnit;
```

extract_type

The mandatory parameter `extract_type` specifies how a `select_filter` is to be interpreted.

```
extract_type: (matching | subtree);
```

Where:

- `matching` – the Printing DUA only extracts the entries within the search region satisfying the filter.
- `subtree` – the Printing DUA extracts all entries within the search region satisfying the filter along with all entries between an entry satisfying the filter and the base entry. A value of `subtree` is not allowed if the hierarchy is not to be preserved – that is, `preserve_hierarchy` is `no`.

For example: `extract_type : subtree;`

preserve_hierarchy

The mandatory parameter `preserve_hierarchy` specifies whether the hierarchical relationships between the extracted entries are to be preserved.

```
preserve_hierarchy: ( yes | no );
```

If set to `no`, the extracted entries are treated as a set of unrelated entries.

For example: `preserve_hierarchy : yes;`

Entry ordering parameters

If the entry-extraction parameter `preserve_hierarchy` is set to `yes`, the entries are extracted in `preorder`. An order may be specified for each set of entries with the same parent in the tree formed by the base entry and the extracted entries.

If the entry-extraction parameter `preserve_hierarchy` is set to `no`, the extracted entries are treated as a set of unrelated entries and an ordering may be specified. The following parameters specify ordering.

`class_order`

The optional parameter `class_order` specifies the order of the object classes of the extracted entries. If the hierarchy is preserved (i.e. `preserve_hierarchy` is `yes`, then sorting by object class occurs for each set of entries which have the same parent in the tree formed by the base entry and the extracted entries. If no class order is specified then sorting by object class does not occur. Its argument is a list of object class names.

```
class_order: objectclass {, objectclass };
```

For example:

```
class_order : organizationalPerson, organizationalUnit;
```

`entry_sort`

The optional parameter `entry_sort` specifies the sorting criteria for the extracted entries. If `class_order` was specified then this specifies the order of the entries within each object class. If the hierarchy is preserved (i.e. `preserve_hierarchy` is `yes`, then sorting occurs for each set of entries which have the same parent in the tree formed by the base entry and the extracted entries.

The parameter is specified as a list of attribute type information for each object class. The first attribute type is the primary sort key, the second the secondary, etc.

```
entry_sort: class_sort { class_sort };
```

`class_sort` is a specification of the sorting criteria for a particular object class. It has the form:

```
class_sort ::= [ objectclass : fieldspec {, fieldspec } ] ;
```

`fieldspec` is used to specify the attribute types to sort on. This may be an attribute in the current entry represented by *type*, or it may be an attribute in another entry referred to by a `DistinguishedName` value stored in an attribute in the current entry. This is represented by `type.type` where the first *type* must be an attribute in the current entry with a syntax of `DistinguishedName` and the second *type* is an attribute in the referenced entry.

```
fieldspec ::= type | type . type ;
```

Where an attribute used as a sort key has multiple values then only the first value retrieved is used for the sort. Attributes of a particular type are sorted in ascending order according to the ordering rule for that type.

For example:

```
entry_sort :
    [ organizationalPerson : sortSubs, surname, givenName ]
    [ organizationalRole : sortSubs, roleOccupant.surname,
      roleOccupant.givenName ]      ;
```

Data output parameters

These parameters specify the data to be extracted from the selected entries and how this data should be outputted.

file_start

The optional parameter `file_start` specifies the text string to be placed at the beginning of the file.

```
file_start: string;
```

For example:

```
file_start : "Current list of units\n\n";
```

entry_output

The mandatory parameter `entry_output` specifies the data to be output and the way in which it is to be written. Each object class to be output must be represented by an output specification which describes how an entry belonging to the object class is to be output. The syntax is

```
entry_output: objectclassoutput { objectclassoutput } ;
objectclassoutput ::= [ objectclass [+] :
    entrystart , { { datastart , datatype , dataend } } ,
    entryend ]
datatype ::= typespec [+] [ ( number ) ] |
    typespec [+] [ ( [ rdnSelect , ] number ) ]
    | subordinate | path
```

The items `subordinate` and `path` are described on page 133.

```
typespec ::= type [ . componentspec ] |
    type . type [ . componentspec ]
componentspec ::= component { . component }
```

`typespec` is used to identify the attribute type to display. This may be an attribute in the current entry or it may be an attribute in an entry referenced by a `DistinguishedName` value in an attribute of the current entry. For example, to display the `surname` attribute of the entry pointed to by the `roleOccupant` attribute of an `organizationalRole` object class, the following `objectclassoutput` could be used:

```
[ organizationalRole: "", roleOccupant.surname, "" ]
```

`componentspec` is an optional extra information which may be used to extract fields from complex attribute syntaxes. e.g. to print out the `telephoneNumber` field in a `facsimileTelephoneNumber`, the following `typespec` could be used:

```
facsimileTelephoneNumber.telephoneNumber
```

The *component* may be used to access a named field in structure (as above), or it may be a number providing an index into a SEQUENCE OF or SET OF list where the first element in the list has an index of 1. Using an index value of zero will cause the number of elements in the list to be output. Using a negative index will reference elements from the end of the list – for example, -1 will reference the last element in the list.

`entrystart`, `datastart`, `dataend` and `entryend` are each defined as *string*. Each of these must be present even if empty. If the `#` character occurs in one of the strings, it is replaced with a number corresponding to the depth of the entry i.e. the distance between the entry and the base entry. This special meaning can be overridden by preceding the `#` with the escape character `\`.

If `datastart` or `dataend` begins with a `*`, the `*` is not output but indicates that the remainder of the string should be output even if the associated attribute is not present in the entry.

There are two ways to output attributes with multiple values: either only the 'first' value is output or else all the values are output. The output of multi-valued attributes is controlled by the following:

- if a `+` is appended to the object class name then all values of the listed attributes will be output,
- if a `+` is appended to the name of an attribute then all the values of that attribute will be output,
- if an attribute is specified more than once for a particular object class then all the values of that attribute will be output,
- if none of the above apply for a particular attribute then only the 'first' value will be output.

An attribute may be specified more than once for a particular object class to cater for the attribute having more than one value. Where an attribute is specified more than once, all the specifications must be contiguous. If an attribute to be output has two values then the second value may be output in a different way to the first value by a second output specification for the attribute. If a second specification is not present then the second occurrence of the value is output in the same way as the first. In general, if an attribute to be output has n values then the k th value, where $1 < k = n$, is output as specified in the k th occurrence of the attribute in the output statement, or if such a statement is not present, it is output as specified in the last occurrence of the attribute in the output statement.

If an attribute to be output has attribute syntax of either `BOOLEAN` or `DistinguishedName` then it may be accompanied by a set of arguments which control the presentation of the value(s). These arguments are described later in the chapter.

For example:

```
entry_output :
    [ organizationalUnit : "*# ",
      {"", organizationalUnitName, ""}
      {"\nMgr: ", manager, ""}
      , "\n\n"]
    ;
```

file_end

The optional parameter `file_end` specifies the text strings to be placed at the end of the file.

```
file_end: string;
```

data_formats

The optional parameter `data_formats` contains a list of formatting instructions for attributes with attribute syntax `BOOLEAN`, attributes with attribute syntax `DistinguishedName` and for *path* information. The syntax is

```
data_formats: format { format };
```

```
format ::= formatBool | formatDN | formatSubordinate
```

Each `format` item in the list contains a complete formatting instruction.

A `formatBool` item is a formatting instruction for attributes with attribute syntax `BOOLEAN`.

A `formatDN` item is a formatting instruction for attributes with attribute syntax `DistinguishedName` and for *path* information. A `formatSubordinate` item is a formatting instruction for subordinate object classes. The first argument in each formatting instruction is the reference number of the item. This is the value used in the `entry_output` parameter to refer to a particular format.

formatBool

The syntax for `formatBool` is

```
[ number : TRUEstring , FALSEstring ]
```

The items `TRUEstring` and `FALSEstring` are each a *string*.

`TRUEstring` specifies the string to be output if the value is `TRUE`; `FALSEstring` specifies the string to be output if the value is `FALSE`. If a `BOOLEAN` value is to be output and no formatting instruction is given, these values default to "Y" and "N" respectively.

formatDN

The syntax for `formatDN` is

```
[ number : [ compress , ] [ reverse , ]  
          sepRDN , startRDN , endRDN, sepAVA [ , sepTV ] ]
```

The items `sepRDN`, `startRDN`, `endRDN`, `sepAVA` and `sepTV` are each a *string*.

The second argument indicates whether or not the extracted information is to be abbreviated. If `compress` is specified then the information will be abbreviated; otherwise, it will be output normally. Only values of type `organizationalUnitName` may be abbreviated.

The third argument indicates whether the extracted RDNs are to be output in normal order (with the first RDN being the closest to the root) or in reverse order (with the first RDN being furthest from the root). If `reverse` is specified then the RDNs are output in reverse order; otherwise they are output in normal order.

`sepRDN` specifies a text string to separate the RDNs. In the case where an RDN has more than one AVA, `startRDN` specifies a text string to be placed on the left hand side of the RDN, `endRDN` a text string to be placed on the right hand side, and `sepAVA` a text string to separate the AVAs. The final argument `sepTV` specifies a text string to separate the type and value strings in each AVA. This last argument is optional - if it is omitted then *only the attribute value* of each AVA will be output. If a DN or *path* value is to be output and no formatting instruction is given, these values default to " , " , " (" , ") " , " , " and "=" respectively.

The syntax for `formatSubordinate` is:

```
formatSubordinate ::= [ number : formatSubClass { ,
                      formatSubClass } ]
formatSubClass ::= { object_class : start_string , end_string }
```

The `start_string` is printed before the first subordinate with object class `object_class` of the current entry and the `end_string` is printed after the last subordinate with the object class `object_class` of the current entry. The `start_string` and `end_string` are only printed if the current entry has at least one subordinate of the defined object class. A `formatSubordinate` data format is only used in conjunction with a `%subordinates typespec`.

For example:

```
data_formats : [ 1: " | ", "", "", "" ]
               [ 2: "YES", "NO" ] ;
               [ 3: { organizationalUnit: "<UNIT>", "</UNIT>" },
                   { organizationalRole: "<ROLE>", "</ROLE>" } ]
```

Subordinates

The `%subordinates` ‘pseudo-attribute’ may be used as a place holder to indicate where the current entries’ subordinate information should be printed relative to the attribute information of the entry. If not present, the subordinate information is printed after the `entryend` string of the entry.

The `%subordinates` directive may have an optional format identifier which may be used to reference a `data_format` which may be used to specify extra formatting information for each subordinate object class.

```
subordinates ::= %subordinates [ ( number ) ]
```

Path data

In addition to attribute information, information from an entry’s Distinguished Name may also be output. This is accomplished by the definition of the “pseudo-attribute” `%path`. `%path` represents the Distinguished Name of the entry being written to output. It permits the entry’s Distinguished Name to be treated as if it were an attribute.

`%path` may be specified any number of times for a particular object class in the `entry_output` parameter. Unlike the case for ordinary attributes, multiple instances do not have to be contiguous. Also there is no special meaning attached to the instances after the first one; each instance is processed individually.

Distinguished Name output

The output of distinguished names (i.e. attributes which have attribute syntax of `DistinguishedName` and of `path` information) is controlled by a pair of optional arguments.

An attributes which has attribute syntax of `DistinguishedName` is specified as follows in the `entry_output` parameter:

```
type [+] [ ( [ rdnSelect , ] number ) ]
```

`Path` information is specified as follows in the `entry_output` parameter:

```
%path [ ( [ rdnSelect , ] number ) ]
```

The first argument, *rdnSelect* indicates the RDNs to be extracted. It has the form:

```
rdnSelect ::= all | selector | selector selector [ _ [ - ]
```

```
number ] ]
```

```
selector ::= number | $ [ - number ] | @ [ + number ]
```

A value *all* indicates that all the RDNs in the distinguished name are to be extracted. A value *k* where *k* is an integer indicates that the *k*th RDN is to be extracted, where $1 \leq k \leq (\text{the number of RDNs in the distinguished name})$. A value of '\$' indicates the last RDN while a value of "\$ - *k*" where $1 \leq k \leq (\text{the number of RDNs in the distinguished name} - 1)$ indicates the RDN at distance *k* from the last RDN. If the entry represented by the distinguished name being processed is a subordinate of the base entry then a value of "@ " indicates the RDN of the base entry while a value of "@ + *k*" where $1 \leq k \leq (\text{the number of RDNs in the distinguished name} - \text{the number of RDNs in the distinguished name of the base entry})$ indicates the RDN at distance *k* beyond the RDN of the base entry.

If two selectors are entered as the value then they specify a subset of the distinguished name. If the distinguished name is considered as a sequence of RDNs with the first RDN being the one closest to the DIT root, then the subset is the contiguous subsequence of RDNs starting at the RDN represented by the first selector and ending at the RDN represented by the second i.e. the selectors define the end points of the subsequence. If for an entry the RDN representing the first selector is further from the DIT root than the RDN representing the second then the subsequence is null. If the selectors in *rdnSelect* are not accompanied by a *number* in parentheses then the subsequence defined by them is extracted. Otherwise, a maximum of *number* RDNs are extracted where *number* is a non-zero positive integer. If *number* is not preceded by a minus sign then only the first *number* RDNs of the subsequence are extracted (if there are less than *number* RDNs in the subsequence then they are all extracted). If *number* is preceded by a minus sign then only the last *number* RDNs of the subsequence are extracted.

If the entry represented by the distinguished name being processed is not a subordinate of the base entry and a selector of either "@" or "@+*k*" has been specified, then the following action is taken:

- The entire *rdnSelect* argument is ignored i.e. all the RDNs in the distinguished name are extracted;
- A warning message is placed in the error file.

The message is:

```
WARNING - attribute xx for following entry has distinguished  
name value outside of selected range: dd
```

where *xx* is the name of the attribute which possesses the distinguished name value and *dd* is the Distinguished Name of the entry which has the attribute.

NOTE: The above condition can only arise in the processing of an attribute which has attribute syntax of DistinguishedName. It cannot arise in the processing of %path.

If the value of *rdnSelect* is a single selector which is not in the allowable range then no data is extracted.

If two selectors are specified and one or both of the selectors are outside the allowable range then:

- If the second selector is k where $k > (\text{the number of RDNs in the distinguished name})$ or is $@ + k$ where $k > (\text{the number of RDNs in the distinguished name} - \text{the number of RDNs in the distinguished name of the base entry})$ then it is processed as if it were "\$". If the first selector has this property then no RDNs are extracted.
- If the first selector is "\$ - k " where $k > (\text{the number of RDNs in the distinguished name} - 1)$ then it is processed as if it were "1". If the second selector has this property then no RDNs are extracted.

The second argument controlling the output of a distinguished name is a *number* indicating which formatting statement in `data_formats` is to be used for formatting the extracted data. This argument must be present.

The first argument, *rdnSelect* is optional. If it is not provided then the entire distinguished name is extracted. If both arguments are omitted, the entire distinguished name is output using the default formatting.

For example:

```
%path(@+1 $-1 (2), 1)
```

Parameter-value definitions

string

A *string* is any sequence of characters enclosed in matching single or double quotation marks. The opening and closing quotes of the string must be on the same line. If the string contains a quote of the same type as the enclosing quotes then the embedded quote must be escaped by duplicating it in the string. For example, the string value O'Hara could be quoted as 'O"Hara' (*the " is two single quotes*) or "O'Hara".

Note that text strings specified in the script may contain non-printable characters as well as printable ones. Non-printable characters are represented using a backslash followed by either a special character or a three-digit octal code for the character. The supported special characters are `\n` (linefeed), `\t` (tab), `\\` (backslash), `\'` (single quote), and `\"` (double quote).

number

A *number* is a sequence of digits.

objectclass

An *objectclass* is the name of an object class (either X.500 or View500 built-in, or a user-defined name specified via an `objectClasses` operational attribute), or an object identifier. An object identifier may be optionally preceded by the keyword `class`.

For example:

```
organizationalUnit
deltawingOrgPerson
{1 3 32 0 1 6 23}
```

Supported attribute syntaxes

The Printing DUA converts attributes with a complex syntax into simple text strings. It supports the following subset of the attribute syntaxes supported by the DSA:

BOOLEAN

DirectoryString

DistinguishedName

FacsimileTelephoneNumber

GeneralizedTime

INTEGER

NumericString

ORAddress

ORName

PostalAddress

PrintableString

TeletexString

TelexNumber

UTCTime

Chapter 8

Printing DUA scripts

ViewDS includes several scripts for extracting information from the demonstration directory, Deltawing. These scripts can be adapted for any other directory and serve as a starting point for developing new scripts.

Scripts

The following are supplied scripts:

| | |
|-------------------------------|---|
| <code>phonelist.ds</code> | Simple phone list for an organizational unit giving name, unit and phone number for all staff in the unit. |
| <code>unitlist.ds</code> | List of organizational units, with address, phone and fax number. |
| <code>executivelist.ds</code> | List of all persons whose job title begins with 'A/C Executive' with name, phone and job title. |
| <code>mailinglist.ds</code> | List of all persons whose mailing address does not include the word 'Australia', with name, title and mailing address. |
| <code>staffdetails.ds</code> | List of all persons with most of their attributes, and all fields tagged (for importing into a desktop publishing package). |

phonelist.ds

This script produces a simple phone list giving the name, unit and phone number for each person in an organizational unit.

Input script

```
# This script produces a phone list. The name, and phone number
# of each person are given as well as the units they belong to.

base :
    organizationName "Deltawing"
    / organizationalUnitName "Deltawing Information Systems
Ltd."
    / organizationalUnitName "Home Media Division"
    ;
```

```

# This parameter has no role in this case. Setting is
# arbitrary.

include_base : no ;

# Select all people
select_filter :
    objectClass = organizationalPerson ;

# Extract only entries matching filter
extract_type : matching ;

# Don't preserve hierarchical relationships between extracted
# entries
preserve_hierarchy : no ;

# Sort by Surname, Given name, Telephone number
entry_sort :
    [ organizationalPerson : surname, givenName,
    telephoneNumber ]
    ;

# Header on output file
file_start : "Phone List\n\n" ;

# Data output
entry_output :
# Output instructions for each person
    [ organizationalPerson : "",
# Surname
        { "", surname, ""}
# Comma then space then given name then tab
        { ", ", givenName, *"\t"}
# Two units immediately after "base", then tab
        { "", %path(@+1 $-1 (2), 1), *"\t"}
# All telephone numbers - separated by comma space
        { "", telephoneNumber, ""}
        { ", ", telephoneNumber, ""}
# Terminate line
        , "\n" ]
    ;

# Output format of the two superior units selected above
data_formats :
# Compress the names of the units and separate them with comma
    [ 1: compress, ",", "", "", "" ]
    ;

```

Sample output of phonelist.ds

...

| | | |
|------------------|-----------------|----------------|
| Briggs, Robert | HomeEd | (03) 9335 7132 |
| Brkic, John | HomeEd,MarkRes | (03) 9335 3145 |
| Bruce, Martina | WWWS,StratRelat | (03) 9335 4647 |
| Budgen, Ronald | HomeEd,MarkRes | (03) 9335 3131 |
| Campbell, Rodney | HomeEd,MarkRes | (03) 9335 8968 |
| Cassio, Suzy | HomeEd,MarkRes | (03) 9335 8882 |
| Chan, Tim | HomeEd,MarkRes | (03) 9335 8356 |
| Cheep, Carol | WWWS,WebDev | (03) 9335 4224 |
| Clifford, Robert | HomeEd,MarkRes | (03) 9335 3167 |
| Colenutt, Gary | HomeEd,MarkRes | (03) 9335 3177 |
| Corbet, Meredith | WWWS | (03) 9335 4526 |
| Crompton, Peter | HomeEd,MarkRes | (03) 9335 1435 |
| Crowhurst, Derek | HomeEd,MarkRes | (03) 9335 3967 |
| Cullen, Greg | HomeEd,MarkRes | (03) 9335 6780 |
| Dang, Em | HomeEd,MarkRes | (03) 9335 8720 |
| ... | | |

unitlist.ds

This script produces a list of units in hierarchical order together with the location and telephone and fax numbers for each unit.

Input script

```
# This script produces a list of units in hierarchical order.
# The location, telephone number and facsimile telephone
# number of each unit are also output.

# Specify base unit
base :
    organizationName "Deltawing"
    / organizationalUnitName "Deltawing Automotive Ltd."
    / organizationalUnitName "Avalon Factory"
    ;

# Include base unit in search
include_base : yes;

# Select all units
select_filter : objectClass = organizationalUnit;

# Extract entries satisfying above filter together with the
# units
# they belong to
extract_type : subtree;

# list units in hierarchical order
preserve_hierarchy : yes;
```

```

# Units with the same parent are to be sorted in the same way
as in
# the on-line system
entry_sort :
    [ organizationalUnit : SortSubs, OrganizationalUnitName]
    ;

# Header on output file
file_start : "Current list of units\n\n";

# Data output
entry_output :
# Output instructions for each unit ...
# Asterisk followed by "level" of unit followed by space
    [ organizationalUnit : "*# ",
# Unit name
        {"", organizationalUnitName, ""}
# Linebreak followed by label followed by location
        {"\nAdd: ", location, ""}
# Linebreak followed by label followed by telephone number
        {"\nTel: ", telephoneNumber, ""}
# Linebreak followed by label followed by facsimile telephone
number
        {"\nFax: ", facsimileTelephoneNumber, ""}
# Terminate the current line and follow with a blank line
        , "\n\n"]
    ;

```

Sample output of unitlist.ds

```

...
*1 Human Resources Group
Add: 74 Deltawing Road One, Lara, Victoria, 3212
Tel: (052) 35 5882
Fax: (052) 35 6121

*1 Production
Add: 74 Deltawing Road One, Lara, Victoria, 3212
Tel: (052) 35 6119
Fax: (052) 35 2178

*2 Operations
Add: 74 Deltawing Road One, Lara, Victoria, 3122
Tel: (052) 35 6934
Fax: (052) 35 2178
*3 Operations Support
Add: 74 Deltawing Road One, Lara, Victoria, 3122
Tel: (052) 35 4541
Fax: (052) 35 4593

```

...

executivelist.ds

This script produces a list of all persons whose job title begins with 'A/C Executive' with name, phone and job title.

Input Script

```
# This script produces a list of A/C Executives.

base :
    organizationName "Deltawing"
    ;

# This parameter has no role in this case. Setting is
arbitrary.
include_base : no ;

# Select all people with title "A/C Executive"
select_filter :
    objectClass = organizationalPerson and title ~ "A/C
Executive"
    ;

# Extract only entries matching filter
extract_type : matching ;

# No hierarchical relationships between extracted entries
preserve_hierarchy : no ;

# Sort by telephone number, surname then given name
entry_sort :
    [ organizationalPerson : telephoneNumber, surname,
givenName ]
    ;

# Header on output file
file_start : "List of A/C Executives\n" ;

# Data output
entry_output :
# Output instructions for each person ...
    [ organizationalPerson : "",
# Attributes separated by tabs
        { "", telephoneNumber, *"\t"}
        { "", surname, *"\t"}
        { "", givenName, *"\t"}
        { "", title, ""}
# Terminate line
```

```
, "\n" ]
;
```

Sample output of executivelist.ds

```
...
(07) 362 8090      Bonnett   Gloria   A/C Executive : Far North
Queensland
(07) 362 8090      Micallef  Alfred   A/C Executive : Northern NSW
(09) 445 5034      Mariebel  Carole   Sales Executive : South
Australia
(09) 445 5055      Balmzian  Noshik   Sales Executive : Western
Australia
(09) 445 5071      BrownRoxanna A/C Executive : Perth Region
+1-212-667-9577    PintoCristino A/C Executive : North East
+1-212-667-9578    CostaCarmello A/C Executive : North West
...
```

mailinglist.ds

This script produces a list of all persons whose mailing address does not include the word 'Australia', with name, title and mailing address.

Input script

```
# This script produces a mailing list for the library.

base :
    organizationName "Deltawing"
    /organizationalUnitName "Deltawing Automotive Ltd."
    ;

# Omit base
include_base : no ;

# Select all overseas staff
select_filter:
    objectClass = organizationalPerson and
        not ( mailingAddress = * "Australia" * )
    ;

# Extract only entries matching filter
extract_type : matching;

# Don't preserve hierarchical relationships between extracted
entries
preserve_hierarchy : no ;

# Data output
entry_output :
```



```
# Output instructions for each staff
# Blank line
    [ organizationalPerson : "\n",
# Attributes in the form of label followed by space followed by
value
    { *"Name: ", givenName, *" "}
    { *"", surname, *"\n"}
    { *"  ", title, *"\n"}
    { *"Post: ", mailingAddress, *"\n"}
# Terminate line
    , "\n" ]
;
```

Sample output of mailinglist.ds

```
...
Name: Carmello Costa
    A/C Executive : North West
Post: Deltawing USA Ltd, 115 Fifth Avenue, New York, NY 10003

Name: Veronica Brennan
    Sales Administration : California
Post: Deltawing USA Ltd, 115 Fifth Avenue, New York, NY 10003

Name: Marie Gander
    A/C Executive : South West
Post: Deltawing USA Ltd, 115 Fifth Avenue, New York, NY 10003
...
```

staffdetails.ds

This script produces a list of all persons with most of their attributes, and all fields tagged (for importing into a desktop publishing package).

Input script

```
# This script outputs most attributes for units and people.
base :
    organizationName "Deltawing"
    / organizationalUnitName "Deltawing Information Systems Ltd."
    / organizationalUnitName "Applications Development"
;

# Include base
include_base : yes ;
exclude_subtrees:
# Exclude entries which are not units or people and all entries
# under them
    not (objectClass = organizationalUnit
        or objectClass = organizationalPerson)
;
```

```

# Setting is arbitrary in this case as a subtree is extracted
# anyway
extract_type : subtree ;

# Preserve hierarchical relationships between extracted entries
preserve_hierarchy : yes ;

# Where extracted entries have the same parent unit, output
# people first
# then units.
class_order : organizationalPerson, organizationalUnit;

entry_sort :
# Units with the same parent are to be sorted in the same way
# as in
# the on-line system
    [ organizationalUnit : sortSubs, organizationalUnitName ]
# People with the same parent are to be sorted in the same way
# as in
# the on-line system
    [ organizationalPerson : sortSubs, surname, givenName ]
    ;

# Data output
entry_output :
# Output instructions for each unit ...
# Line break
    [ organizationalUnit : "\n",
# Tag (backslash followed by "level" of unit followed by U)
# followed
# by unit name
    { *"\#U", organizationalUnitName, ""}
# All attribute values output as tab followed by tag followed
# by
# attribute value
    { "\t\\L", location+, ""}
    { "\t\\T", telephoneNumber+, ""}
    { "\t\\F", facsimileTelephoneNumber+, ""}
    { "\t\\K", keylinkAddress+, ""}
    { "\t\\E", emailAddress+, ""}
    { "\t\\X", telexNumber+, ""}
    { "\t\\Y", teletexTerminalIdentifier+, ""}
# Terminate line
    , "\n" ]
# Output instructions for each person ...
    [ organizationalPerson : "",
# Common name
    { *"\#C", commonName, ""}
# All attribute values output as tab followed by tag followed

```

```
# by attribute value
{ "\t\\S", surname, ""}
{ "\t\\G", givenName+, ""}
{ "\t\\L", location+, ""}
{ "\t\\T", telephoneNumber+, ""}
{ "\t\\F", facsimileTelephoneNumber+, ""}
{ "\t\\K", keylinkAddress+, ""}
{ "\t\\E", emailAddress+, ""}
{ "\t\\X", telexNumber+, ""}
{ "\t\\O", mobileNumber+, ""}
{ "\t\\P", pagerNumber+, ""}
{ "\t\\Y", teletexTerminalIdentifier+, ""}
# Terminate line
, "\n" ]
;
```

Sample output of staffdetails.ds

```
...
\CFrank Nitzsche \SNitzsche \GFrank \L28th Floor, 74 King
Street, Melbourne, Victoria, 3000 \T(03) 9335 8654\F(03) 9335
7800
\CAnna Palozzi \SPalozzi \GAnna \L28th Floor, 74 King Street,
Melbourne, Victoria, 3000 \T(03) 9335 8534 \F(03) 9335
7800
\CBarbara Ryan \SRyan \GBarbara \L28th Floor, 74 King Street,
Melbourne, Victoria, 3000 \T(03) 9335 8514 \F(03) 9335
7800
\CEnnio Torresan \STorresan \GEnnio \L28th Floor, 74 King
Street, Melbourne, Victoria, 3000 \T(03) 9335 1423\F(03) 9335
7800
\CJacqueline Turner \STurner \GJacqueline \L28th Floor, 74
King Street, Melbourne, Victoria, 3000 \T(03) 9335 8760
\F(03) 9335 7800
```

Chapter 9

Advanced features

This chapter describes several advanced Access Presence features, and includes the steps to configure for them.

It includes the following:

- Configuring proxy authorization for ‘single sign on’
- Configuring related-entry workflow
- Configuring the approval process

Configuring proxy authorization for ‘single sign on’

Proxy authorization facilitates ‘single sign on’ – that is, when a user logs on to their computer, through Windows for example, they also log onto ViewDS automatically. This streamlines ‘self service’ and there is no need to maintain a separate set of passwords for ViewDS users.

This chapter has four sections:

- What is proxy authorization
- How proxy authorization works
- Requirements
- Implementing proxy authorization

What is proxy authorization

Proxy authorization is a process where ViewDS assigns privileges based on logon information passed to it by an external agent, rather than its own authentication mechanism.

The external agent is usually a web server, which passes the logon information to Access Presence through an environment variable that uniquely identifies an entry in the directory. The name of the environment variable is set by the configuration-file parameter `webRemoteUser` (see page 18) which by default is `REMOTE_USER`.

The proxy authorization process uses mechanisms defined in *IETF Standard RFC 3875 – The Common Gateway Interface (CGI) Version 1.1*; and *IETF Standard RFC 4370 – Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control*.

How proxy authorization works

Figure 5 provides an overview of how proxy authorization works.

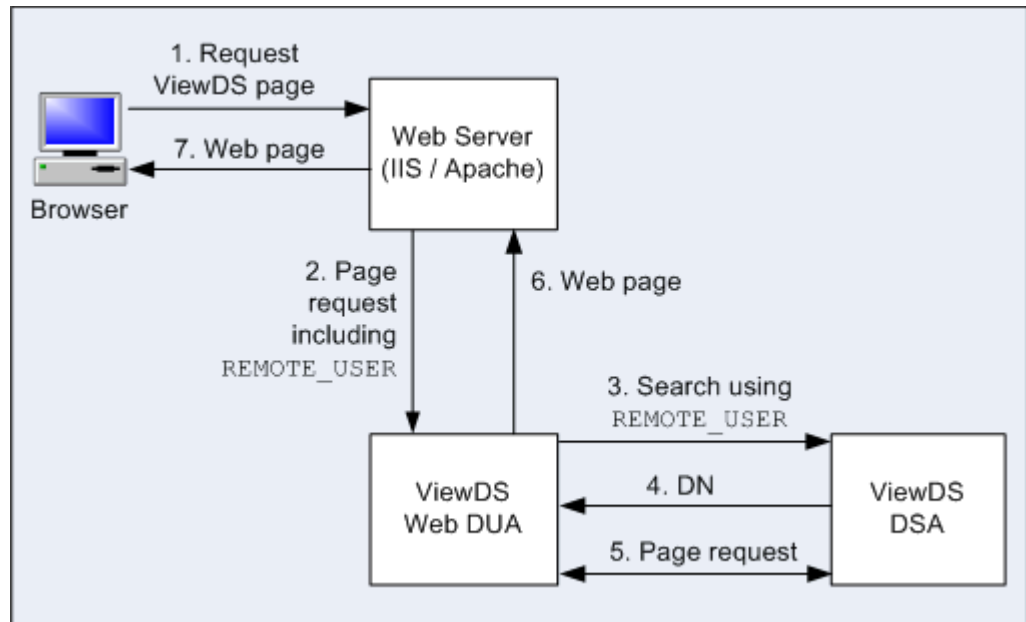


Figure 5: How proxy authorization works

The steps in Figure 5 are described below:

1. The user logs onto Windows and requests an Access Presence page.
The web server recognises that the requested page requires authentication, and communicates with the client computer to confirm who is logged on.
The choice of authentication mechanism is a configuration option of the web server. Possible options include SSPI (Windows authentication), Netegrity SiteMinder or HTTP basic authentication.
2. The web server sends the page request to Access Presence, and sets the environment variable `REMOTE_USER` to the user's system-authentication ID.
3. Access Presence connects to the directory as the user that has been designated the 'proxy user'. It then searches the directory for the entry whose *authorization attribute* matches the value of `REMOTE_USER`. (An attribute is designated the authorization attribute during configuration for proxy authorization. It must be set to the user's system-authentication ID.)
4. If the search matches a single entry, the DSA returns the user's Distinguished Name (DN). Otherwise, the DSA returns nothing and Access Presence uses an empty DN to represent an anonymous user.
5. Access Presence communicates with the DSA to obtain information for the requested page. (This is restricted according to the privileges of the DN obtained in the previous step.)
6. Access Presence returns the resulting web page to the web server.
7. The web server passes the page to the browser.

Requirements

There are several requirements for proxy authorization:

- An authorization attribute must be selected to identify each directory user by their system-authentication ID.
- The web server used with Access Presence must support an authentication process and be able to publish the system-authentication ID to the environment variable `REMOTE_USER`.
- The browser used must support the same authentication mechanism as that used in the web server.
- To allow 'self service', your installation of ViewDS must implement Basic Access Control (BAC).

Implementing proxy authorization

Implementing proxy authorization involves modifying the ViewDS configuration file (see page 15) and web server used by Access Presence:

1. Select an authorization attribute to identify each directory user by their system-authentication ID.
2. Set the configuration-file parameter `webProxyUser` to `on`. (If the parameter is not included in the configuration file, add it.)

```
webProxyUser = on
```
3. Set the configuration-file parameter `webProxyAuthAttribute` to the name of the authorization attribute (the attribute you selected to store system-authentication IDs).
 For example:

```
webProxyAuthAttribute = uid
```
4. Set each authorization attribute to the appropriate system-authentication ID. For example, with Active Directory, the value is `<domain>\<samAccountName>`.
5. Create a directory user to be the 'proxy user'.
6. Add the operational attribute `proxyAgent` to the entry for the 'proxy user' (see the help topic *Add an attribute to an entry*).
7. Set the configuration-file parameter `webProxyUser` to the user name and password of the 'proxy user'.
 For example:

```
webBindUser = vfsuper passwd
```
8. Set up your web server for authentication.
 For Microsoft IIS, enable any of the authentication modes for the Access Presence pages. For Apache, set up SSPI authentication for example.

Configuring related-entry workflow

An organization might have a policy that allows multiple entries for the same person. The policy might allow one actual entry for the person and several references to this entry were each represents, for example, a different role the person fulfills.

For example, consider a policy that involves `organizationalPerson` and `organizationalRole` entries. It allows each employee to have just one `organizationalPerson` entry, but this can be referenced by several `organizationalRole` entries to represent the employee's roles in the organization. This is illustrated below.

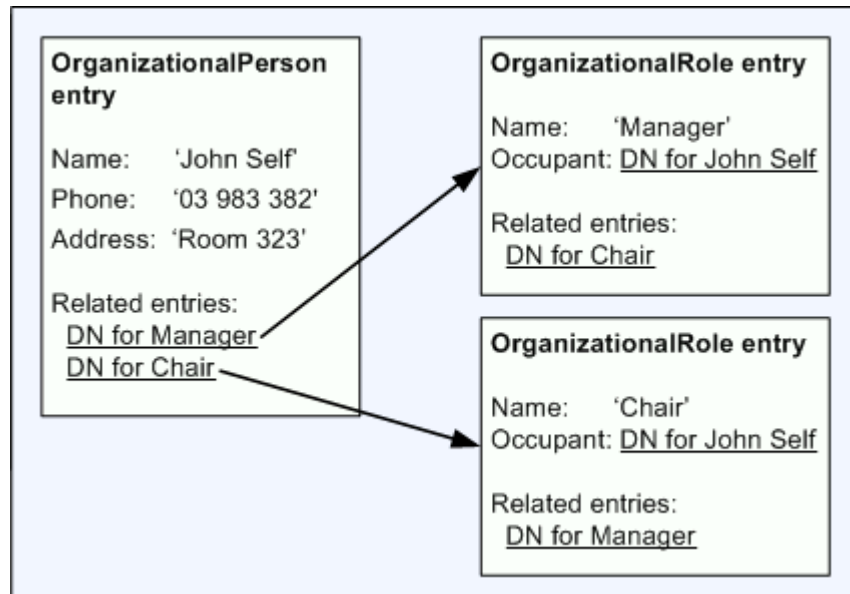


Figure 6: Related entries

The related-entry workflow encourages a user to comply with the required policy when they create a new entry for someone who already has an entry. When a user adds a new entry for the above `organizationalPerson`, the workflow is as follows:

1. Access Presence presents the user with a Search Form (the particular Search Form can be specified).
2. The user searches for 'John Self'.
3. The Search Results page (specific to the related-entry workflow) lists all entries that fit the search criteria, and displays a button next to each that allows the user to create a new `organizationalRole` to reference the entry.
The page also gives the user the option to create a new `organizationalPerson` entry if there is no existing appropriate entry.
4. The user creates a new `organizationalRole` for an entry. Access Presence automatically populates the new entry's attributes with the user's search criteria (so they don't have to enter the same information twice).
5. Access Presence adds a related-entry link to the new `organizationalRole` entry, to all existing `organizationalRole` entries, and to the `organizationalPerson` entry.

Implementing the related-entry workflow

1. Optionally, define a new Search Form to be used in the related-entry workflow. You can define a Search Form through the ViewDS Management Agent (see the help task *Create a Search Form*).
2. From the ViewDS Management Agent, perform the help task *Define related entries* for the required object class. (In the above example, related entries would be defined for the `organizationalPerson` object class.)
3. Optionally, if a template should be used instead of the Search Form template during the related-entries workflow, declare the template in the configuration-file parameter `webRelatedSearchTemplate` (see page 19).
4. Optionally, if a template should be used instead of the Search Results template during the related-entries workflow, declare the template in the configuration-file parameter `webRelatedSearchResultTemplate` (see page 20).

Configuring the approval process

This mechanism imposes an approval process on changes to the directory.

A user is designated either a 'requestor' or an 'approver'. A requestor can submit a request to add, modify, delete or move an entry. Later, a user with appropriate access rights, an approver, can either approve or reject the request.

Using the approval process

The approval process applies when a requestor modifies, deletes, moves or adds a subordinate to an entry. The following three example scenarios describe the approval process from a user's perspective.

Request modify

1. A requestor views an entry on the Expanded Entry page, which includes the 'Request modify' link (see `VFRequestModifyHref` on page 50). Note that this link is only displayed to requestors.
2. The requestor clicks the 'Request modify' link. The Modify page is displayed, which contains the 'Request reason' box and the attributes that can be modified by a requestor.
3. The requestor enters the new attribute values and the reason for their request, and submits the request.
4. Later, an approver is presented with a list of pending requests (see `VFRequestList` on page 51).
5. The approver clicks the request and the Modify page displays the requestor's changes and the reason for the modification. The page also includes 'Reject' and 'Approve' buttons (the default presentation of these buttons can be modified through the format file – see the *Approval process tags* on page 82).

Request single entry delete

1. A requestor views an entry on the Expanded Entry page.
2. The requestor clicks the 'Request delete' link (see `VFRequestDeleteHref` on page 50). The Request Remove Entry page is displayed (see page 60) which contains the 'Request reason' box.
3. The requestor enters their reason and submits the request.
4. Later, an approver is presented with a list of pending requests (see `VFRequestList` on page 51).
5. The approver clicks the request and the Request Remove Entry page is displayed, this time showing the reason for the request along with the 'Reject' and 'Approve' buttons.

Request move

1. A requestor adds several entries to the target-object cache and then views another entry on the Expanded Entry page.
2. The requestor clicks the 'Request move' link (see `VFRequestMoveHref` on page 50). The Select Target Object page (see page 64) page displays the entries in the target-object cache that are valid possible superiors to the entry, along with the 'Request reason' box.
3. The requestor enters their reason, selects a new superior and then submits the request.
4. Later, an approver is presented with a list of pending requests (see `VFRequestList` on page 51).
5. The approver clicks the request and the Select Target Object page is displayed, this time showing:
 - the reason for the request
 - the 'Reject' and 'Approve' buttons
 - the list of valid possible superiors, but with the requestor's choice selected

Implementing the approval process

To implement the approval process:

1. Select the attributes that requestors can modify.
2. Assign users to be requestors or approvers.
3. Create and modify the appropriate templates.

These steps are discussed below.

1. Select the attributes that requestors can modify

To specify the attributes that can be modified by requestors, perform the following ViewDS Management Agent help topic: *Select the attributes that requestors can modify*.

2. Assign users to be requestors or approvers

Users are assigned as requestors or approvers through either Basic Access Control or ViewDS Access Control.

If you are using Basic Access Control

Allocate the following access rights (see the ViewDS Management Agent help topic *View or modify an Access Control Item*):

- *Requestors* should have modify access to the operational attribute `viewDSUpdateRequest`, but not to the attributes available for modification.
- *Approvers* should have modify access to the operational attribute `viewDSUpdateRequest` and also to the attributes available for modification.

If you are using ViewDS Access Control

Allocate the following access rights (see the ViewDS Management Agent help topic *Set the ViewDS Access Control for an entry*):

- *Requestors* should have the `accessLevel` of `requestor`.
- *Approvers* should have the `accessLevel` of `updater`, `admin` or `superuser`.

3. Create and modify the appropriate templates

Make the following changes:

- Add the *Approval process tags* (see page 50) to the Expanded Entry page.
- Add an `InputChecked` directive to the format file (see page 71).
- Create a Request Remove Entry template (see page 60) and add the configuration-file parameter `webRequestRemoveEntryTemplate` (see page 24).