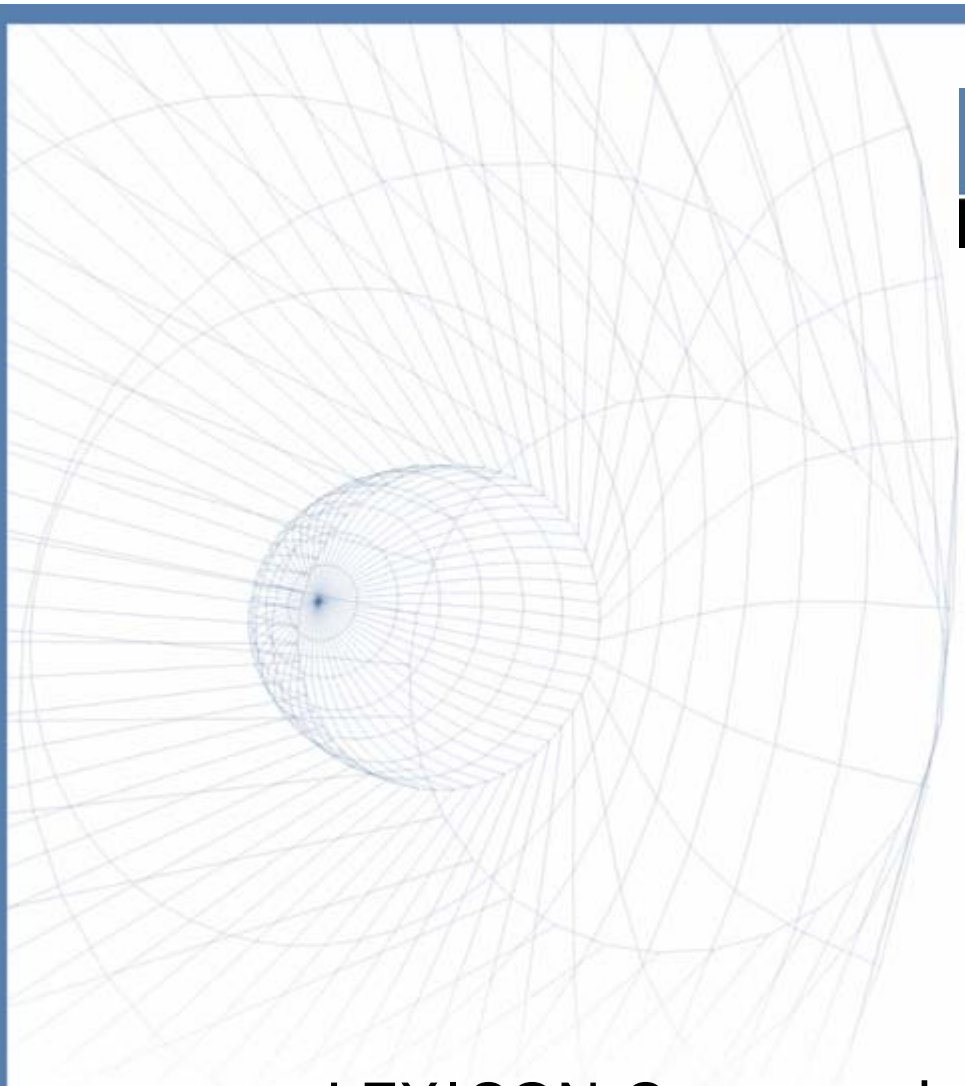


LEXICON Command Reference Manual

www.aveva.com

AVEVA Solutions Ltd

Disclaimer

Information of a technical nature, and particulars of the product and its use, is given by AVEVA Solutions Ltd and its subsidiaries without warranty. AVEVA Solutions Ltd and its subsidiaries disclaim any and all warranties and conditions, expressed or implied, to the fullest extent permitted by law.

Neither the author nor AVEVA Solutions Ltd, or any of its subsidiaries, shall be liable to any person or entity for any actions, claims, loss or damage arising from the use or possession of any information, particulars, or errors in this publication, or any incorrect use of the product, whatsoever.

Copyright

First published September 2007

AVEVA Solutions Ltd, High Cross, Madingley Road, Cambridge, CB3 0HB, United Kingdom

Trademarks

LEXICON Reference Manual

Contents Page

LEXICON Command

1 Introducing LEXICON

1.1 Who Should Read This Manual

This Reference Manual assumes that you are familiar with using OUTFITTING at a System Administrator level. Access to LEXICON is normally restricted to FREE users only.

1.2 What Does LEXICON Do?

The LEXICON enables the OUTFITTING system administrator to create their own attributes and elements within a project.

There are three separate feature sets available to the administrator which are covered in this document, these are UDA (User Defined Attributes), UDETs (User Defined Element Types) and USDAs (User System Defined Attributes). Once created in the LEXICON they can be added to the databases of a project.

1.3 LEXICON Database

Before a project can use any of the resources covered in this manual a DICTIONARY (DICT) database must be created as part of a Multiple Data Base (MDB). This is done by using the ADMIN module in the same way as for any other type of database. For further information refer to the *Adminstrator Reference Manual*.

Elements within the LEXICON may be changed or deleted during the course of a project. The dictionary will contain all the definitions for a project and if data is to be transferred between OUTFITTING projects the dictionary must be transferred as well. The data output and project reconfiguration functions can be used to do this. A user may create elements within other databases such as OUTFITTING DESIGN from definitions within the LEXICON database, however if the LEXICON database is removed then these elements will become unrecognised in the OUTFITTING DESIGN database.

2 User Defined Attributes

User Defined Attributes (UDAs) enable the OUTFITTING system administrator to add new attributes to any element in the databases of a project. These UDAs are created as elements in the LEXICON database inside the OUTFITTING project. Because LEXICON databases are project-specific, it is possible to define attributes to suit individual project requirements or company standards.

Once defined, UDAs may be accessed in much the same way as normal attributes, including setting values, querying and reporting upon these values. Utilities such as the Data Output and Project Reconfiguration functions treat UDAs as they would any other attribute. UDAs can also be used in expressions.

The hierarchy of elements which make up a UDA within the LEXICON database is shown below.

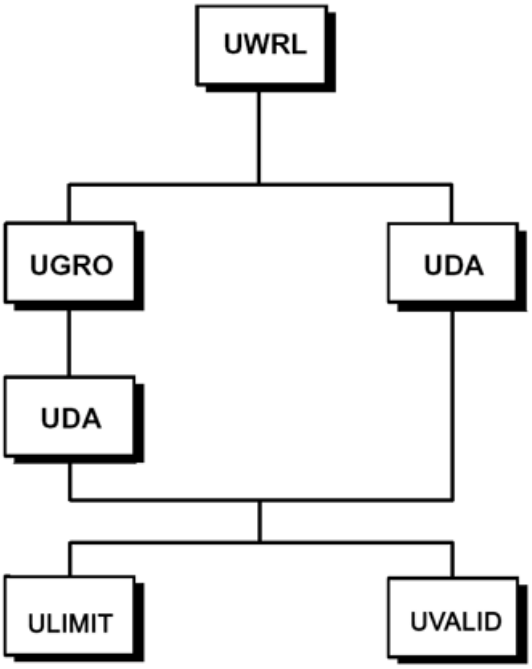


Figure 2:1. The UDA Hierarchy

UWRL (UDA World) is the top-level administrative element. As well as the standard OUTFITTING attributes, this has a **DESC** (description) attribute which can contain up to 120 characters of text.

A UWRL may hold **UGRO** (UDA Group) elements or individual **UDA** (UDA definition) elements. A UGRO has the same attributes as a UWRL.

Note: Both UWRL and UGRO elements may themselves be given user-defined attributes.

2.1 UDA Attributes

UDA elements have the following attributes (besides the standard ones):

UDNAME - the name by which the UDA will be referred to in the OUTFITTING modules that use UDAs. This should not be confused with the standard **NAME** attribute of the UDA, which is the name of the UDA element within the LEXICON database. The name must be entered as a text attribute with a maximum length of twenty case-independent characters. An example of setting the UDNAME attribute would be:

```
UDNA 'SUPPORT'
```

ABLEN is the minimum number of characters for a UDNAME abbreviation - default 3. In the default case, the minimum abbreviation for the example UDNAME given above would be SUP. You may not give two or more UDAs the same minimum abbreviation - an error will occur when the UDAs are **compiled** see *Creating New UDAs*.

UKEY. This is a system attribute. It can be queried but not set directly. It is the number which identifies the UDA and provides the link with the UDA instance in the constructor database. The UKEY is generated from the UDNAME name by taking the ASCII values of the first four characters, subtracting 32 (or 64 if lowercase) and multiplying them together. If the first four characters are not unique, an increment is applied to the calculated value until a unique value is found.

If you intend to include foreign databases or merge projects, you should ensure that the UDNAMEs in each database correspond, and that they are unique. This will ensure that the UKEYs generated are the same.

When you compile a Dictionary database, you should include all Dictionary databases for the project in the MDB. This will ensure that any UKEY clashes are resolved when the database is compiled. Alternatively, you could use different naming conventions in different databases.

Note: that changing the UDNAME will not affect the key.

It is possible to copy a UDA, with the copy having the same key as the original - see KEYCOPY command in *Copying UDA Definitions*.

UTYPE is the UDA type. This can be set to the following:

`REAL`	Any numeric value, e.g. `23.5`
`INTeger`	Any whole number, e.g. `200`
`REFerence`	An element identifier, e.g. `/VESS1, =12/99`
`TEXT`	Any characters in closing single quotes. e.g. 'BOT_STEEL2'
`WORD`	Any sequence of letters, e.g. `ABCD`
`LOGical`	`TRUE` or `FALSE`
`POSition`	A position, e.g. `E10 N50 U100`

DIRection	A direction, e.g. `E 45 U`
ORIentation	An orientation, e.g. `U is E and N is D`.

The UTYPE defines the type of data that may be entered against the UDA for an element accessed by a user module. POS, DIR and ORI types are particularly useful when used in conjunction with WRT or FROM. For example:

```
Q :POS OF /EQUI-1 WRT /*
```

Note: when WRT is used, the base value is relative to the Owner, not the element itself. Hence this command will extract the values of :POS for EQUI-1, and concatenate the values with the values for the Zone and Site. The result will be relative to the World.

ULEN is the maximum number of occurrences of the given UTYPE that may be stored in the UDA. For example, a REAL UDA might be given a ULEN of 3, making it suitable for storing, say, a 3D coordinate array. For TEXT and WORD UTYPEs, ULEN is the maximum number of characters. The default setting of ULEN is 1, the maximum is 120. (Strings longer than 120 characters will be truncated.)

RPTX (Report text) is a text attribute used to define the default column heading to be used in reports. RPTX may be up to 20 characters long.

UUNIT (UDA units) is used to modify the value contained by the UDA such that it is output in the units specified by the user. For example, if you set distance units to be feet and inches using the FINCH DIST command in OUTFITTING DESIGN, the value of the UDA will be output in these units (provided also that UUNIT is set to DIST; see below).

UUNIT is a word attribute. The available word options are BORE, DISTANCE (minimum abbreviation DIST) or NONE. The default is NONE, which means that the UDA value will be output exactly as it is stored and will not vary according to BORE or DIST unit settings. UUNIT should be set to DIST for POS type UDAs.

The UUNIT setting is only used if the UDA type is REAL

DFLT (Default). For all UDA types except REF, a default setting may be specified. For UDA types INT, REAL or LOG the default will only be taken if ULEN is 1. DFLT is not a valid attribute for POS, ORI or DIR types.

The type of setting entered should correspond to the UTYPE. DFLT may be left unset or set to NONE. In this case the UDA has no default.

DESCRIPT is a text attribute which may be used to give a description of the UDA.

ELELIST (Element list) contains the list of element types for which the UDA is valid. The elements may exist in the OUTFITTING DESIGN, CATALOGUE or OUTFITTING DRAFT (PADD) databases, or in the LEXICON database itself. Up to 100 element types may be set.

The element list must be assembled using the syntax:

```
ELElist ADD element_type
```

Elements must be removed from the list by using the syntax

```
ELElist REMove element_type...
```

Example:

```
ELELIST ADD ZONE        add ZONE and PIPE elements
PIPE

ELE REMOVE PIPE         remove PIPE elements

ELE REM ALL             remove all elements
```

To simplify input the keywords ALLP and PRIM can be used to cover all piping components and all primitives, respectively.

UDAs can be binded to UDETs.

UDAs may be added to the UDET in the same way that they are allocated to system element types, i.e. a UDET may appear in the ELELIST for a UDA definition.

REFLIST (Reference list). This is only relevant if UTYPE is REF. It is the list of valid element types that the UDA may reference. For example, if a REF UDA is set in OUTFITTING DESIGN to an EQUI element and the UDA element's REFLIST is set to VALV, then an error will result. If REFLIST is not set, it is assumed that all element types may be referenced. REFLIST is set in a similar way to ELELIST, for example:

Example:

```
REFLIST REM PIPE BRAN

REF REM ALL

REF REM PRIM
```

The REFLIST may have up to 100 members.

A RELIST may also contain references to UDETs which have been directly derived from a system type.

UHIDE Setting this to True will indicate that the UDA will be hidden from the 'Q ATT' command and from the attribute form within OUTFITTING. Querying of the individual UDA will not be effected by this setting.

UTEAMS If the UHIDE attribute has been set to true then the UTEAMS attribute can be used to override UHIDE for specific teams. The UDA will then be hidden for all teams other than the ones specifically declared in UTEAMS. UTEAMS can hold an array of values, for example:

Example:

```
UTEAMS /*TEAM1/*TEAM2/*TEAM3
```

Values may also be input individually for example:

Example:

```
UTEAMS /*TEAM1
UTEAMS NUM 2/*TEAM2
UTEAMS NUM 3/*TEAM3
```

UHYPER This attribute holds a logical value, setting this to the true will allow the text value of a UDA to contain a path to the external file.

UCONN This attribute holds a logical value, setting the value to have will signal that the UDA is a connection in the reference list.

UPSEUD Setting this logical attribute to True will indicate the UDA is pseudo attribute. Pseudo attribute allow for dynamic values to be returned as needed rather than having static values stored in the database.

The code required to return a pseudo attribute value must be plugged in through the use of the C# API.

UDPEND If the calculation of the pseudo attributes relies on other values on the element, and the pseudo attribute is to be used in a dynamic rule, then it is possible to denote what real attributes the pseudo attribute depends on by setting the UDPEND attribute. For example:

Add a UDA :VOLUME to a box. The provided pseudo code simply multiplies XLEN*YLEN*ZLEN, so :VOLUME is dependant on XLEN, YLEN, ZLEN. This should be indicated by setting UDPEND attribute to 'XLEN YELN ZLEN'. If subsequently, :VOLUME is included in a dynamic rule, and XLEN, YLEN, ZLEN are modified then the result will reflect this. For most pseudo attributes the setting of UDPEND is unlikely to be needed.

2.2 Valid Values

The valid values for a text or word UDA can be defined by creating UVALID elements below a UDA.

The attribute of a UVALID element is:

UVVAL - valid value.

There may be UVALID elements before a UDA.

If the administration requires multiple validation criteria then multiple elements must be created. The following example will create a UDA which will allow values "Warning", "Danger", or "Caution":

Example:

```
NEW UDA /STAT UTYP TEXT          Create a new Text UDA

UDNA 'STATUS'

ULEN 10
```

```
RPTX 'STATUS'

ELE ADD EQUI STRU SUBS PIPE
ZONE SITE

NEW UVALID /WARNING          Create a UVALID element to hold a condition

UVVAL ' WARNING'

DESC 'Condition Warning'

/STAT                        Navigate back to UDA element

NEW UVALID /DANGER           Create a second UVALID element

UVVAL 'DANGER'

DESC 'Condition Danger'

/STAT

NEW UVALID /CAUTION          Create a third UVALID element

UVVAL 'CAUTION'

DESC 'Condition Caution'
```

2.3 Limits

The limits for a real or integer UDA can be defined by creating ULIMIT elements below a UDA.

The attributes of a ULIMIT element are:-

UMIN - Minimum value

UMAX - Maximum value

The following example will create a Weight UDA and limit the input value to between 5 and 50 :

Example:

```
NEW UDA /WEIG UTYP REAL

UDNA 'WEIGHT'

ULEN 3

UUNI DIST

RPTX 'Weight'

ELE ADD STRU SUBS EQUI       Create UDA Weight
```

NEW ULIMIT	Create a ULIMIT element beneath UDA
UMIN 5	Set a minimum value of 5
UMAX 50	Set a maximum value of 50
DESC ‘Weight Limit’	

3 Creating and Using UDAs

3.1 Creating New UDAs

The first step is to create a LEXICON database hierarchy, as shown in *Figure 2:1.: The UDA Hierarchy*. Commands used to create, delete and navigate around the LEXICON database elements are exactly the same as those used for any other database type: see the *Software Customisation Guide* for details.

A typical LEXICON session could be as follows:

Example:

```
NEW UWRL /UWRL1

NEW UGRO /CENTRE_GRAVITY

DESC 'Centre of gravity at tributes'      Create UDA World, UDA Group

NEW UDA /LENGTH UTYP REAL                 Create and name UDA, type REAL

UDNA 'LENGTH'                             Assign 'user database' attribute name,

ABLEN 3                                   Minimum abbreviation LEN

UUNI DIST                                 Set UDA units as system DISTANCE units

DFLT 0.00                                 Set default UDA value
```

Here, the default value of the UDA has been set to zero, but in a case such as temperature it might be set to be a value of (say) 20.0 to represent an ambient temperature.

```
RPTX 'Length'                 Set column heading to be used in reports

ELE ADD STRU SUBS             Define element types that are to use the UDA

COMPILE                       Implement the UDA definition
```

The COMPILE command (minimum abbreviation COMP, no arguments) causes the UDA to be made available for use by the named element types in the appropriate database. New UDAs, or changes to existing UDAs, will not be accessible in user databases unless the COMPILE command is used. It is not strictly necessary to use the COMPILE command if a UDA is deleted, but this is advisable.

Compiled changes will not be implemented if you leave LEXICON by using the QUIT command.

3.2 Copying UDA Definitions

When a UDA definition is copied, a new **key** (a reference number known only to the system) is assigned to the new element. Occasionally you may want to have more than one definition with the same key; for example, to vary the default or UDA length depending on the element types. To do this the key may be copied using the command

KEYCOPY *name*

rather than the standard

COPY *name*

Where two or more definitions exist with the same key, the definitions **must** have the same name, minimum abbreviation and type, and **different** ELELISTS, otherwise an error will occur on compilation. It is advisable to have the same units and RPTX text. Note that project reconfiguration will always copy the key.

3.3 Example UDAs

This section gives examples of some of the different UDA types. The commands for creating each UDA type are given, together with a brief discussion of their features.

Note: When a UDA appears in the attribute list of a user element, its name (i.e. its UDNAME) will be preceded by a colon (:) to distinguish it as a UDA.

Example:

UDA type REAL

```
NEW UDA /WEIG UTYP REAL

UDNA 'WEIGHT'

ULEN 3

UUNI DIST

RPTX 'Weight'

ELE ADD STRU SUBS EQUI
```

This is an example of a REAL attribute which could be used to store the weight of an element.

Example:

UDA type TEXT

```
NEW UDA /CHKD UTYP TEXT

UDNA 'CHKD'
```

Example:

```
ULEN 6

RPTX 'Checked By'

ELE ADD EQUI STRU SUBS PIPE ZONE SITE
```

This is an example of a TEXT attribute which could be used to store the name of the person who has checked a particular element. After checking, the checker would set the attribute to his name.

Note the disparity between the length of the text string (ULEN) and the report text (RPTX). In this case, the text length is six characters for the attribute, but the report text is set as being 10 characters long. The Reporting utility will set its column width to whichever is the largest; there is little point in having only six characters for the attribute when 10 characters will be allowed anyway. Conversely, if six characters are all that is required, then the RPTX should be cut down to reduce the column width.

Example:

UDA type LOGICAL

```
NEW UDA /TEST UTYP LOGICAL

UDNA 'TESTED'

ABLEN 4

DFLT FALSE

RPTX 'TESTED'

ELE ADD BRAN EQUI
```

This is an example of a LOGICAL attribute which is used to store either TRUE or FALSE to indicate one of two conditions. In this case, the attribute is intended to indicate whether an item has been pressure tested or not. ABLEN allows the attribute to be accessed as either :TEST, :TESTE or :TESTED.

Example:

UDA type REFERENCE

```
NEW UDA /SUPPORT UTYP REFERENCE

UDNA 'SUPPORT'

ULEN 50

RPTX 'Support reference'

ELE ADD ELBO TEE REDU FLAN VALV CROS CAP ATTA PCOM
```

Example:

```
ELE ADD OLET INST

REF ADD STRU SUBS EQUI
```

This is an example of a REFERENCE attribute which allows a number of piping components to be referenced to Structural items or Equipment. As the name suggests, this attribute could be used to indicate the name of a Structural element which is supporting a Component. The attribute would be set in the database in the same way as other reference attributes, e.g. :SUPP /COL-A1.

Note that the REFLIST attribute has been set such that the UDA in the user element (i.e. the :SUP attribute) can only be set to an identifier relating to a Structure, a Substructure or an Equipment.

If the referenced element is renamed or deleted, the contents of the :SUP attribute in the user element(s) will be updated accordingly.

Example:

UDA type WORD

```
NEW UDA /OPERATION UTYP WORD

UDNA 'OPERATION'

RPTX 'Normal Operation'

ELE ADD VALV
```

This example shows how a WORD attribute can be set up to show the normal state of Valves; for example, whether they are normally open or normally closed. The value assigned to this attribute in the OUTFITTING DESIGN would probably be either OPEN or CLOS. For this particular example, a default value is inappropriate, so the DFLT attribute is left unset and no default is shown in the OUTFITTING DESIGN. As with other unset attributes, the default text '---' will be output in reports whenever an attribute with no default value is encountered.

Example:

UDA type POSITION

```
NEW UDA /COFG UTYP POS

UDNA 'COFG'

ULEN 1

UUNIT DIST

RPTX 'Centre of gravity'

ELE ADD STRU SUBS EQUI
```

This is an example of a POSITION attribute which could be used to store the position of the centre of gravity of an element. Because position is always a three-field array, the ULEN of this attribute is set to 1. Moving the element using AT or BY commands will not change the values stored in COFG, unless they are linked by a Rule. For example:

```
:COFG DYNAM (N 100 WRT CE)
```

Then, if the element is moved :COFG is reset automatically.

Example:

UDA type ORIENTATION

```
NEW UDA /HAND UTYP ORI

UDNA 'HANDWHEEL'

ULEN 1

RPTX 'Handwheel'

ELE ADD VALV
```

This is an example of a ORIENTATION attribute which could be used to store the orientation of an element. Because orientation is always a three-field array, the ULEN of this attribute is set to 1.

Example:

UDA type DIRECTION

```
NEW UDA /ACCESS UTYP DIR

UDNA 'ACCS'

ULEN 1

RPTX 'Access'

ELE ADD STRU SUBS EQUI
```

This is an example of a POSITION attribute which could be used to store the direction of an element. Because direction is always a three-field array, the ULEN of this attribute is set to 1.

3.4 UDAs in User Elements

The remaining sections of this chapter discuss various aspects of UDAs in the user modules.

3.4.1 Introduction

UDAs are applicable to all database types.

Having defined and compiled a set of UDAs in LEXICON, those elements defined as using UDAs will have the UDA UDNAME, preceded by a colon and followed by the UDA's default setting, added to their standard attribute lists. For example, suppose an INTEGER UDA has been defined as follows:

```
Name /INT
Lock false
Owner /UGRO1
Udname 'CAPACITY'
Ablen 3
Utype INT
Ulen 1
Rptx text
Dflt 80
Descript text
Elelist SITE
Reflist
```

Querying the attributes of a SITE element in OUTFITTING DESIGN would give something like:

```
Name /SITE1
Lock false
Owner /*
Position E 0mm  N 0mm  U 0mm
Orient
      Y is N
and   Z is U
:CAP 80
```

The UDA **instance**, that is, the reference to the UDA definition, appears at the end of the attribute list, with a colon as its first character.

3.4.2 Setting and Changing UDAs in User Elements

A UDA instance is set using the syntax

:Udname setting

where *setting* could be an integer, a value, text, etc., depending on the UDA type. An example of (re)setting the CAP attribute above could be:

```
:CAP 101  - reset CAP value to 160
```

Array attributes may be set or changed as shown by the following examples. Changes are shown relative to the original setting.

Example:

`:ARR1 1 2 3 4 5`	Set to 1 2 3 4 5
`:ARR1 NUM 4 44`	Change 4th value to '44', i.e. change to 1 2 3 44 5
`:ARR1 NUM 5 55 66`	Change '5' to '55 66' (i.e. change to 1 2 3 4 55 66)

No more values can be specified than are given by the UDA length (ULEN attribute). If more values are entered, the excess ones are ignored and an error message is output.

An attribute may be returned to its default state by a command such as

```
:ARR2 DEF
```

If no default exists, the UDA becomes unset. This always applies to REFERENCE, POS, ORI and DIR UDA types since no defaults are allowed. REFERENCE UDAs may only reference elements of the type given in the UDA definition.

3.4.3 Querying UDAs in User Elements

Querying the attributes of an element will return the UDA name followed by its value(s). The UDAs appear at the end of the attribute list, each UDA name being preceded by a colon.

UDAs can be queried individually by a command such as

```
Q :UDA1
```

(the value(s) of the UDA will be returned).

For array UDAs, individual array values may be queried by commands such as

```
Q :UDA [1]  Query array value 1
```

The UDA **definition** may be queried using a command such as

```
Q UDADEF :UDA1
```

The UDA name, length, description, minimum abbreviation length, defaults, units and list of valid element types will be output.

3.5 Use of GOTO

It is possible to go to a UDA reference as with other references. For reference arrays the command may be qualified by the NUMBER keyword. For example,

Example:

`GOTO :REF2 [4}`	Go to the 4th reference in the list for UDA :REF2

3.5.1 Effect on UDA Instances if Definition is Changed

In all that follows it should be remembered that no change to a UDA instance (apart from deletion of the UDA definition element) will occur until the COMPILE command has been given in LEXICON.

Where an instance of a UDA exists, then changes to the definition may cause changes to the instance.

If the UDA type (UTYPE attribute) is altered, the current instances will be inaccessible. Unless UDA instances are **purged** (see next subsection), the process is reversible, i.e. resetting the type will re-enable access to the instances.

If the UDA length (ULEN attribute) is decreased, latter parts of numeric arrays or text strings will become inaccessible. This process is reversible, i.e. if ULEN is subsequently increased, the values may again be accessed.

If an element type is removed from an ELELIST, instances of these element types will be inaccessible. Again, this is reversible.

This indirect alteration of the apparent contents of the database could be disturbing for users and it is recommended that the System Administrator should avoid modifying UDA definitions once they are in use.

3.5.2 Purging UDA Instances

When a UDA definition is deleted, all instances of that UDA will *appear* to be lost from the databases. However, the database space taken up by the UDA instances will not become usable until the command

```
PURGE UDAS
```

has been given from the user module. The space released is then recoverable when the database is next compacted. Instances will not be PURGEd from locked elements.

Note: The PURGE command will operate on any UDA instance that it sees as being invalid. This includes any UDA instance which is defined in a deferred LEXICON database. Making the deferred database current will not result in the UDA instances being recovered.

3.5.3 Expressions and Rules

Note: UDAs can be used in expressions and rules wherever an attribute is appropriate.

3.6 Handling of UDAs in Reports and Expressions

Elements in the LEXICON database can be reported on in the same way as other attributes. UDAs may be used to select elements for reporting and to sort elements by the value of a UDA or of a field of an array attribute. Reference UDAs may be used in the same manner as normal reference attributes to extract data from the referenced element.

The DFLT attribute setting is always regarded as being text for this purpose, so meaningful report output would probably not be obtained if the DFLT setting is not text. However, an expression such as

```
UDA WITH DFLT EQ '10'   (rather than EQ 10)
```

would give meaningful output.

The RPTX attribute is used as the default column heading in the report.

For more information about reports and expressions, see *Reporting from OUTFITTING* and the *Design Software Customisation Guide.*

3.7 Handling of UDAs in Data Output Macros

All or part of a LEXICON database may be listed out using the OUTPUT facility. The resulting macros may be read into LEXICON in the normal way.

As with other attributes, defaults will not appear in the output file unless the DEFAULT option is selected.

When using OUTPUT with DESI, PADD or CATA databases that contain UDAs, the data file will fail when read into a OUTFITTING DESIGN/Drawing/CATALOGUE module if it contains UDA instances whose definitions are not available on the target project or if the definition is inconsistent with this instance.

3.8 Handling of UDAs in Project Reconfiguration

Project reconfiguration from ADMIN may be used in the normal way for transferring LEXICON database data. After being reconfigured, the LEXICON database will be compiled automatically. OUTFITTING DESIGN databases do not need to be included in the update if a LEXICON database is reconfigured. The normal use of reconfiguration to update projects is unaffected; all UDA instances will be preserved. The space taken up by PURGED UDA instances will be reclaimed.

3.9 Reference External Document

A UDAs can refer to an external document. Setting the UHYPER attribute on a UDA will allow the user to launch the document (using the associated Microsoft Windows application) from the Attribute Utility within the OUTFITTING DESIGN Module.

During a LEXICON session create a UDA as follows.

Example:

```
NEW  UDA  /FILELINK  UTYP   Create a new Text UDA
TEXT

UDNA 'FILELINK'              Make sure the length is enough to accommodate the file path

ULENGTH 50

UHYPER TRUE

ELE  ADD  EQUI  STRU  SUBS
PIPE ZONE SITE
```

Now switch to the OUTFITTING DESIGN and from the command line type the following to set the UDA value.

:FILELINK 'C:\TEST.XLS'

From the Attribute Utility in OUTFITTING DESIGN it is now possible to right click the UDA and open the file.

4 User Defined Element Types

Within OUTFITTING the standard set of elements are often used for a variety of purposes. A UDET allows an element to be created with a more meaningful name, For example an administrator could create clearly defined sub types such as Pipe Lagging, Pipe Painted and Pipe Heated.

The hierarchy of elements which make up a UDET within the LEXICON database is shown below.

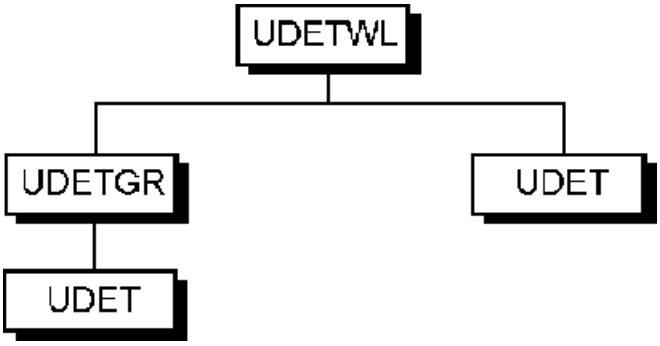


Figure 4:1. The UDET Hierarchy

UDETWL - (UDET World) is a top-level administrative element. As well as the standard OUTFITTING attributes, this has a DESC attribute which can contain up to 120 characters of text.

A UDETWL may hold **UDETGR** (UDET Group) elements or individual **UDET** (UDET definition) elements. A UDETGR has the same attributes as a UDETWL.

4.1 UDETs Attributes

UDETs have the following attributes:

BASETY - The BASETY attribute indicates what system type the UDET is derived from. The UDET will inherit all of the attributes of the base type.

UDNAME - the name by which the UDET will be referred to in the OUTFITTING modules that use UDETs.

UDOLIST - An array listing the allowed owner types within the Databases hierarcy e.g. Site, Zone.

UDMLIST - An array listing the allowed member lists e.g. NBOX, NPOLYH.

UDHLIS - An array listing the system attributes to hide when displaying a UDET.

UDATLS - Read-only Pseudo array containing a list of valid attributes.

4.2 Creating a UDET Definition

The creation of a UDET has many similarities to the creation of a UDA covered in the previous chapters of this document.

UDET definitions must be created in the dictionary database as members of a UDET group (UDETGR) which in turn must exist as members of a UDET world (UDETWL).

A typical LEXICON session could be as follows:

Example:

```
NEW UDETWL /MYUDETWL1

NEW UDETGR /MYUDETGR1

NEW UDET /MYUDET                Create new UDET

UDNA 'TESTUDET'                 Name the new UDET

BASETY EQUIP                    Base the new UDET on an equipment
```

As with UDAs, the UDNAME attribute is used to hold the UDET name. This name will then be used for identification in the constructor modules.

Subsequently in constructor modules, elements of this type will be identified with a prefix colon e.g.

```
:TESTUDET
```

The UDNAME name may be up to 50 characters long. Unlike UDAs there is no minimum abbreviation.

A UDET must be based on any visible element type in the OUTFITTING DESIGN, OUTFITTING DRAFT, CATALOGUE and Property DBs. The BASETY attribute indicates what system type the UDET is derived from.

4.3 Redefine the Allowed Member List

When creating a UDET by default it will inherit the same allowed member types (UDMLIS) and allowed owner types (UDOLIS) as it's base type. These lists may be reset.

The types in the new list must be either:

- a system type allowed on the base type.
- a UDET derived from a system type allowed on the base type.

Based on the previous example create a UDET named MYBOX based on a system type BOX.

At the command line type 'Q UDLIST' to see the current owner list as follows:

```
EQUI STRU PTRA SUBE SUBS TMPL
```

At the command line type 'Q UDMLIST' to see the current member list as follows:

```
NBOX NPOLYH NCYL NSLC NSNO NDIS NCON NPYR NCTO NRTO NXTR NREV
```

Both lists may be completely redefined or the keywords 'ADD' or 'REMOVE' may be used to add or remove particular entries.

The following example shows this functionality in a session:

Example:

UDOLIST SITE ZONE	Define SITE, ZONE as the only valid owners
UDOLIST ADD SITE ZONE	Add SITE, ZONE as valid owners
UDMLIST REMOVE ALL	Remove all allowed member types

The allowed owner list and member list for a UDET may be any system types allowed on the BASETY, plus any UDET based on these system types. In the example for :MYBOX you could not add a ZONE to UDOLIST, or a EQUI to UDMLIST because the system type BOX of which the UDET is based does not allow this.

Another example would be a :MYBOX can only go under a SUBE, and that another UDET named :MYNBOX can only go under :MYBOX or BOX. This is achieved as follows:

For :MYBOX
UDOLIST SUBE UDMLIST :MYNBOX

For :MYNBOX:
UDOLIST :MYBOX BOX

Changing the allowed owners/members for a UDET could invalidate some existing instances. These are reported as warnings in DICE.

4.4 Allow UDETs based on Zones to own Zones and Vice Versa

It is allowable for ZONES to own UDETs based on a ZONE and vice versa. Recursively structures are not allowed.

For example create a UDET :PIPEZONE based on a ZONE, and a UDET :SMALLPIPES also based on a ZONE. Allow :PIPEZONE below a ZONE, and SMALLPIPE below a :PIPEZONE.

Since :PIPEZONE, and :SMALLPIPES are both derived from ZONES the result is a three level hierarchy.

4.5 Changing Allowed References

The valid list of element types that a UDA may point to is set in the REFLIS attribute of a UDA. The list may hold UDETs as well as system types.

For example if a REFLIS on a UDA holds EQUI :PUMP and :PUMP and :VESSEL are derived from a EQUI, then this UDA may point to a EQUI or a :PUMP but not to a :VESSEL

4.5.1 System Reference Attributes

It is not possible to redefine the valid lists for system reference attributes. A system reference attribute may point to any UDET derived from a valid system type. E.g. an HREF on a BRANCH can reference a :MYNOZZLE since this is derived from a NOZZLE.

4.5.2 UDAs

UDAs may be added to the UDET in the same way that they are allocated to system element types, an example would be to use the following command during a session :ELELIST ADD :PUMP

4.6 Hiding System Attributes

The UHDLIS attribute is an array holding a list of system attributes to hide when displaying a UDET. For example to hide iso-related attributes on a UDET

UDHLIST BSTA HWRF TWRF BRLO RLSTOR TSFBR DELDSG

The ADD and REMOVE keyword can be used in association with UDHLIST.

4.7 Deleting a UDET Definition

In a LEXICON session the DELETE command can be used to remove UDETs,

Object instances of this type will then no longer be recognized. They will default back to the original system type. Any extra UDAs assigned to the UDET will also be deleted.

4.8 UDATLS Pseudo Attribute on a UDET

A read only pseudo attribute UDATLS can be queried, this will return a list of all valid attributes for the UDET. In a LEXICON session navigate to a UDET and enter:

Q UDATLS

4.9 Presentation within the Constructor Modules and ADMIN

The same command line instructions for system types are used when dealing with UDETs, for example

NEW :MYPUMP

DELETE :MYPUMP

Constructed names will be built up using the UDET for example BOX 1 of :PUMP 2.

UDETs may also be deleted by specifying their base type, for example the command DELETE EQUl would delete either EQUl elements or UDETs based on EQUI.

Normally syntax which navigates the database must use the exact type. However navigation syntax which traverse the database will allow a UDET to be matched against its base-type -

For example EQUl will also climb to a :PUMP based on EQUI. The following syntax is valid for a UDET of type :PUMP, as well as explicit use of :PUMP:

BOX 1 OF EQUI

EQUI

This also allows a rule such as RPROP WTHK OF SCTN to work on a FITTING under a UDET based on a SCTN. However NEXT EQUI will not identify a UDET of type: PUMP.

4.10 Use of TYPE at the PML Level

The TYPE attribute returns the base-type, for example EQUI, for UDET. A new attribute ACTTYPE returns the derived type, if one. for example a :PUMP. The FULLTYPE attribute returns the derived type such as EQUIPMENT or :PUMP. To get the Base full-type of an element, the attribute FULSYS should be used.

The table below summarizes the functionality:

	Base type	Derived type
Full word, e.g. EQUIPMENT	FULSYS/TYPE	FULL/TYPE
Short word, e.g. EQUI	TYPE	ACTTYP/E

The attribute query Q ATT (and the general attribute form) will include ACTTYPE as well as type for UDET elements. ACTTYPE will be omitted if the element is not a UDET.

4.11 Collections and Expressions

Any attribute query that takes an element type as a qualifier will take a UDET as a qualifier.

For example

Q MCOUNT :PUMP

This will return the number of :PUMPS in the current members list.

Collections (and related expression syntax) will also work with UDETs.

For example

collect all :PUMP for ISITEA.

Collecting on a type, will collect the derived types.

For example

'COLLECT ALL EQUI'

will include :PUMPS as well as EQUI elements.

If just EQUI elements are required, excluding derived UDETS, they may be collected explicitly using the syntax:

VAR !A COLLECT JUST EQU1 FOR IATEST

This applies to existing OUTFITTING DRAFT selection rules, such as those for line-styles. The user should not need to create new line-style definitions to cover any UDETs, unless they wish to distinguish UDETs from their base elements.

Cursor syntax such as ID EQUI @ identify elements by their base type -thus :PUMP elements will also be found. To exclude UDETs, the syntax ID JUST EQUI @ should be used. If the 'JUST' keyword is used, it will apply to the entire list.

4.12 Changing Type on an Instance

Where UDETs have the same base type or where the base types can be changed to one another, then the UDET may be changed on an instance using the existing 'CHANGETYPE' command. Any UDAs not valid on the new type will be lost.

For example currently BENDS can be changed to ELBOWS and vice versa.

Now define a :MYBEND based on a BEND and a :MYELBOW based on an ELBOW. An instance of type :MYBEND, BEND,:MYELBOW or ELBOW can now be changed to any of BEND, :BEND, ELBOW or :MYELBOW.

If the object has members which are of a type disallowed in the new type, then the switch will be disallowed. Similarly it will be disallowed if the new type is not allowed as a member of it's current owner.

4.13 Element UDET Check

A pseudo attribute UDETCHECK (UDETCH for short) is available for each element to determine its UDET status. Some results from a 'QUERY UDETCHECK' command follows:

Not a UDET.

UDET : test defined by element /TestUdet in database TESTMDB/TESTDICT

UDET definition for :TEST was lost.

UDET :test base type CONE does not match element type BOX

4.14 Purge Report on MDB

In addition to element checking the 'PURGE REPORTS UDETS' command performs an undirected UDET integrity check. This reports on any UDET integrity errors that appear in the current MDB. Each element accessible from the WORLD element is checked for consistency with the UDETs defined in the current MDB. Some example output from this command follows:

/testelement1: UDET definition for :TEST was lost.

/testbox1: UDET :test base type CONE does not match element type BOX

4.15 SPEC Selection

The SPEC question TYPE will allow a UDET.

Hence the first level of spec selection will take the UDET into account.

For example if a component is of type :MYLJSE, then the spec system will do the initial selection on :MYLJSE. The spec might look as follows.

Example:

```
HEADING      TYPE     PBOR0   SHOP   DETAIL  MATXT   CMPREF  BLTREF
NAME
DEFAULT
```

```
-           -         -       =
*/20LJSE    :MYLJSE   20.00   TRUE   /DLJSE  /MLJSE =0      /SB20
*/25LJSE    :MYLJSE   25.00   TRUE   /DLJSE  /MLJSE =0      /SB25
*/32LJSE    :MYLJSE   32.00   TRUE   /DLJSE  /MLJSE =0      /SB32
*/40LJSE    :MYLJSE   40.00   TRUE   /DLJSE  /MLJSE =0      /SB40
*/50LJSE    :MYLJSE   50.00   TRUE   /DLJSE  /MLJSE =0      /SB50
*/65LJSE    :MYLJSE   65.00   TRUE   /DLJSE  /MLJSE =0      /SB65
*/80LJSE    :MYLJSE   80.00   TRUE   /DLJSE  /MLJSE =0      /SB80
*/IOOLJSE   :MYLJSE   100.00  TRUE   /DLJSE  /MLJSE =0      /SBl00
```

4.16 GTYP

Related to the SPEC selection, is the GTYPE. The GTYPE is held on the SCOMP. Normally the GTYPE is the same as the TYPE of the design component, but it does not have to be. Dabacon will report an error if it is not.

It is possible to set a GTYPE to a UDET. For example in PARAGON type:

GTYPE :MYLJSE

Dabacon will continue to check that the GTYPE on the catalogue component is the same as the TYPE on the design component.

5 User System Defined Attributes

A USDA (User System Defined Attribute) allows the administrator to place a behaviour on a standard OUTFITTING element. For example limits may be applied to attributes of top level elements within OUTFITTING.

The hierarchy of elements which make up a USDA within the LEXICON database is shown below.

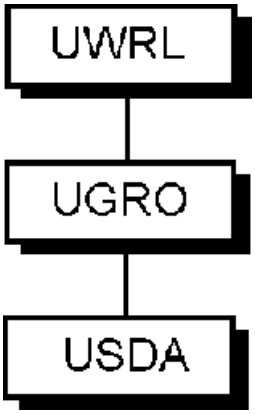


Figure 5:1. The USDA Hierarchy

A USDA allows the administrator to add the following behaviour to system attributes:-

1. Set valid values
2. Define limits
3. Hide attributes on forms
4. Category

The valid values and limits may be varied with element type.

These values are defined by creating a USDA element in the dictionary database. A USDA element has the following attributes:

5.1 USDA Attributes

NAME - the name by which the USDA will be referred to in the OUTFITTING modules.

DESC - is a text attribute which may be used to give a description of the USDA.

USYSTY - underlining system attribute to which the USDA is to be applied.

UALL - logical attribute, if set to true then the USDA will be applied to all OUTFITTING elements with the attribute defined in USYSTY

ELEL - Array containing a list of OUTFITTING elements to apply the USDA to.

UCAT - Allows the administrator to group USDAs on the attributes form within OUTFITTING modules, e.g. `UCAT 'MYCATAGORY'`.

If UCAT is left unset then it has no effect

UHIDE Setting the logical attribute to True will indicate that the USDA will be hidden from the 'Q ATT' command and from the attribute form within OUTFITTING. Querying of the individual UDA will not be effected by this setting.

UTEAMS If the UHIDE attribute has been set then the UTEAMS attribute can be used to indicate what teams the USDA will be available to. UTEAMS can hold an array of values.

5.2 USDA Limits and Valid Values

A USDA may own ULIMIT and UVALID elements to denote the limits and valid values. These are described in *Valid Values* and *Creating a USDA*.

5.3 Creating a USDA

The following example will restrict the FUNC attribute on EQUIPMENT to be 'HeatX' or 'Pump', create a USDA as follows:

Example:

```
NEW UWRL /USDAWR

NEW UGRO /USDAGR

NEW USDA /UFUNC

DESC 'Set restriction on FUNC attribute on EQUI'

USYSTY FUNC

ELELIST ADD EQUI

NEW UVALID

UVVAL 'PUMP'

NEW UVALID

UVVAL 'HEATX'
```

To apply limits to a USDA the procedure is the same as UDAs. The following example will apply limits to the POSSTART attribute on a STANCHION:

Example:

```
NEW USDA /LIMITSTAN
DESC ‘Set limits on POSSTART attribute on SCTN’
USYSTY POSSTART
ELYLIST ADD SCTN
NEW ULIMIT
UMIN 1000
UMAX 6000
```

A Command Syntax Diagrams

A.1 Introduction

This appendix contains the legal command and interrogation syntax diagrams relevant to the OUTFITTING LEXICON module. These diagrams formalise the precise command sequences which may be used and are intended to supplement the explanations given in the appropriate sections of this manual.

A.2 Conventions

The following conventions apply to the syntax diagrams in this appendix:

- All diagrams have abbreviated names. Such names are composed of lowercase letters enclosed in angled brackets, e.g. <assign>. These short names, which are used for cross-referencing purposes in the text and within other syntax diagrams, are supplemented by fuller descriptions where they are not self-explanatory.
- Commands to be input from the terminal are shown in a combination of uppercase and lowercase letters. In general, these commands can be abbreviated; the capital letters indicate the minimum permissible abbreviation. (NOTE: This convention does not mean that the second part of the command must be typed in lowercase letters; commands may be entered in any combination of uppercase and lowercase letters.)

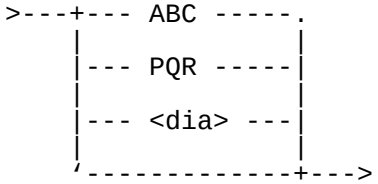
```
ELElist
```

may be input in any of the following forms:

```
ELE
ELEl
ELEli
ELElis
ELElist
```

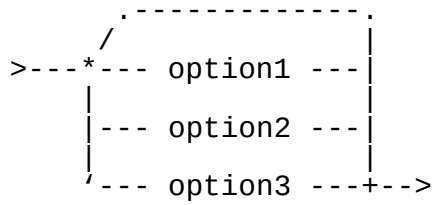
Commands shown *wholly* in uppercase letters cannot be abbreviated.

- Syntax diagrams are generally read from top left to bottom right.
- Points marked with a plus sign (+) are **option junctions** which allow you to input any *one* of the commands to the right of the junction. Thus



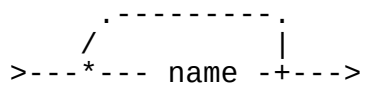
means you may type in ABC *or* PQR *or* any command allowed by the syntax given in diagram <dia> *or* just press RETURN to get the default option.

- Points marked with an asterisk (*) are **loop back junctions**. Command options following these may be repeated as required. Thus



permits any combination of option1 *and/or* option2 *and/or* option3 to be used (where the options may define commands, other syntax diagrams, or command arguments). This may form an exception to the rule of reading from top left to bottom right.

The simplified format



means that you may type in a list of OUTFITTING names, separated by at least one space.

A.3 Command Arguments

These are inputs which are necessary to qualify command words. They are distinguished by appearing in italics.

Name	Definition	Example
name	OUTFITTING element name	/ABCDE
refno	OUTFITTING reference number	=23/1403
integer	a positive integer	0, 3
value	signed number	3.142, -23.66, -34
word	alphabetic word	NULL, VALV (4 chars. max.)
text	alphanumeric string	'Enclose Between Apostrophes'
sign	plus or minus character	+ (for ditto); - (in default lines)
equals	equals character	= (in default lines)
comma	comma character	20, 40 (for range of values)
nl	new line	Press the RETURN key

Table A: 1. Examples of basic command arguments

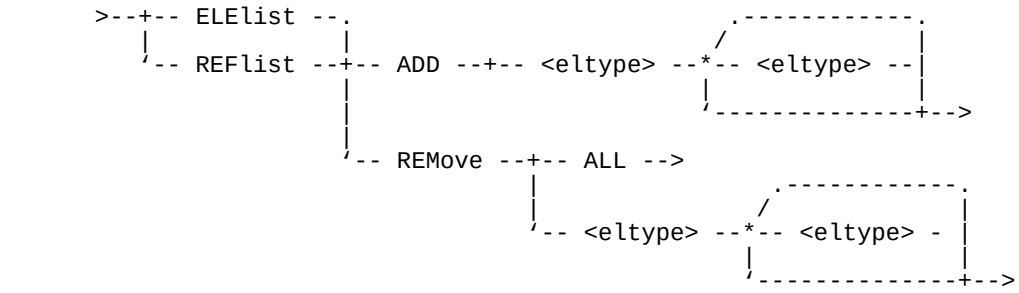
A.4 Syntax Diagrams

Apart from the direct setting of UDA element attributes, the options for which are given in *User Defined Attributes*, the only LEXICON commands are COMPILE (syntax COMPile), ELELIST/REFLIST and COPY/KEYCOPY.

A.4.1 Compiling the Element List and Reference List

See *User Defined Attributes*.

A.4.2 <assign>



<eltype> is any OUTFITTING DESIGN, PADDLE, LEXICON, CATALOGUE or Specification element type: see the appropriate user documentation for lists.

A.4.3 Copying a UDA

See *Copying UDA Definitions*

A.4.4 <ucopy>

```
>--- KEYCopy --- <sgid> --->
```

B Error Messages

This appendix lists those error messages specific to LEXICON. All such messages have a message number beginning with 68. Any other messages that may be output are not described here as they are not specific to LEXICON.

Where messages are self-explanatory no description is included, but the error message is listed for completeness.

(68:3) **The REFLIST ADD command is only valid for a REFERENCE UDA**

You have tried to add an element type to the Reflist of a UDA whose type is not REFERENCE. Check that you have accessed the correct UDA, or reset the Utype attribute, as appropriate.

(68:5) **The element** *element_type* **is already in the ELELIST**

You have tried to add an element type to the ELELIST that is already in the ELELIST.

(68:6) **The element** *element_type* **is not in the ELELIST**

You have tried to remove an element type from the ELELIST that is not in the ELELIST.

(68:7) **The element** *element_type* **is already in the REFLIST**

You have tried to add an element type to the REFLIST that is already in the REFLIST.

(68:8) **The element** *element_type* **is not in the REFLIST**

You have tried to remove an element type from the REFLIST that is not in the REFLIST.

(68:9) **The current element type is not a UDA definition.**

You have tried to modify the ELELIST (or REFLIST) of an element which does not have those attributes.

(68:10) **The ELELIST may only be 100 long.**

(68:11) **The REFLIST may only be 100 long.**

(68:12) **Maximum ABLEN is 20 characters.**

(68:13) **Maximum UDNAME length is 20 characters.**

(68:14) **The UDNAME may not contain a space.**

(68:16) **The ULEN must be between 1 and 120**

(68:17) **The UKEY is set by the system , and may not be set by the user**

(68:18) **WARNING- The UDA definition is not REAL or INTEGER**

You have tried to set DFLT to a real or integer value: you can only do this for a UDA of the corresponding type.

(68:19) **WARNING- The default will only be used if the ULEN is set to 1**

You have set a UDA length (ULEN) which exceeds the length of the default UDA setting (DFLT). Change ULEN (or DFLT).

(68:20) **WARNING- The units will only be used if the UDA type is set to REAL or INTEGER**

You can only set units (UUNI attribute) for a UDA which is of type REAL or INTEGER. Check that you are at the correct UDA or change UTYP, as appropriate.

(68:21) **WARNING- ABLEN is greater than the length of UDNAME**

You have set the minimum abbreviation length of a UDA name to be greater than the length of the UDA name itself. Reset ABLEN or UDNAME as appropriate.

(68:22) **WARNING- The length of UDNAME is less than ABLEN**

(See 68:21)

(68:24) **WARNING- The UDA definition is not a LOGICAL type**

You can only set DFLT to TRUE or FALSE for a UDA which is of type LOGICAL. Check that you are at the correct UDA or change UTYP, as appropriate.

(68:27) **No valid keys remain for new UDA. Please compile dictionary and try again**

No more than 1000 keys may be used.

(68:28) **WARNING- The UDA definition is not a TEXT type**

You can only set DFLT to text for a UDA which is of type TEXT. Check that you are at the correct UDA or change UTYP, as appropriate.

(68:29) **WARNING- The UDA definition is not a WORD type**

You can only set DFLT to a word for a UDA which is of type WORD. Check that you are at the correct UDA or change UTYP, as appropriate.

(68:30) **WARNING- the units text has been truncated to 20 long**

Maximum length of text for the UUNI attribute is 20 characters.

(68:31) **WARNING- BORE and DISTANCE units are only valid for REAL UDAs**

You can only set UUNI to BORE or DIST for a UDA which is of type REAL. Check that you are at the correct UDA or change UTYP, as appropriate.

Note: The following messages will only occur following use of the COMPILE command.

(68:61) **Maximum number of definitions in the dictionary has been exceeded.**

No more than 1000 definitions may be used.

(68:62) **Two UDAs with different keys have the same name** *Udname***. The second definition will be ignored.**

You have given two or more UDAs the same UDNAME.

(68:63) **Two UDAs have the same key (***integer***) but different names or type. The second definition** *Udname* **will be ignored**

(68:64) **Two UDAs with the same key/name (***Udname***) have the same element (***element _type***) in their ELELISTs. Second UDA definition ignored.**

(68:66) **An attempt has been made to assign more UDAs than the maximum limit of 100 to a single element type.**

(68:67) **WARNING for UDA** *name***. The default is not applicable as the ULEN is greater than 1.**

(See 68:19)

(68:68) **WARNING for UDA** *name***. The default type does not correspond to the UDA type. The default has been ignored**

(68:69) **WARNING for UDA** *name***. The UUNIT attribute is not applicable unless a REAL or INTEGER type. It has been ignored.**

(68:70) **WARNING for UDA** *name***. The REFL attribute is not applicable unless a REF type. It has been ignored**

(68:71) **UDA** *name* **has no name and has been ignored**

The named element has no UDNAME.

(68:72) **UDA** *name* **has no element list and has been ignored.**

The ELELIST for the named UDA has no members.

(68:73) **WARNING for UDA** *name***. The ABLEN given is longer than the UDNAME. A value of** *integer* **for ABLEN has been assumed**

(68:74) **Element type** *element type* **has more than 100 UDAs assigned to it. Subsequent UDAs have been lost.**

(68:75) **The maximum number of definitions has been exceeded for the dictionaries in the current MDB**

Maximum number of definitions is 1000.

(68:76) **WARNING for UDA** *name* **BORE, DISTANCE units are only applicable if a REAL UDA type. The units have been ignored.**

(68:77) **UDA** *name* **has no Type set and has been ignored**

You have forgotten to set the UTYPE attribute of the named UDA.

Index