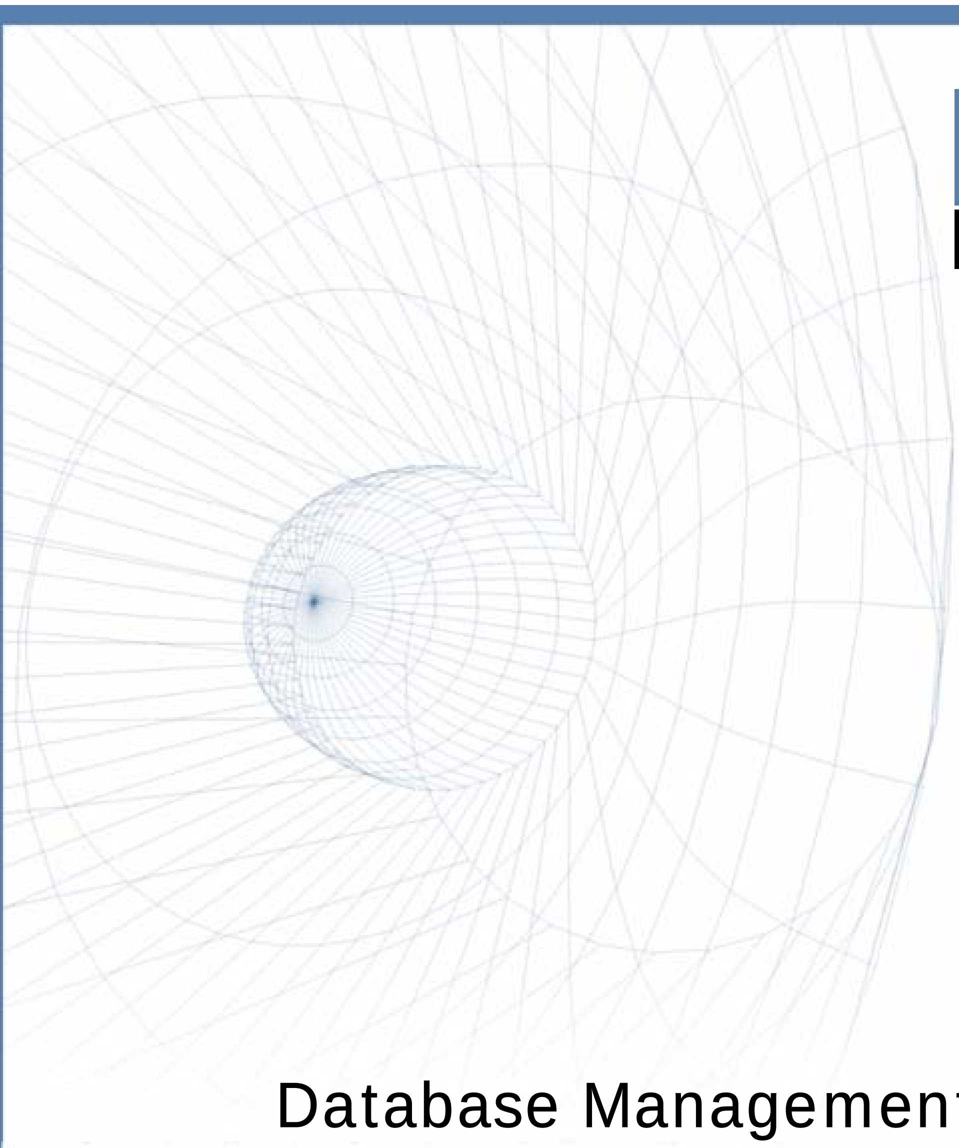
AVEVA

MARINE



Database Management Reference Manual

www.aveva.com

First published September 2007

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

Database Management Reference Manual

Contents **Page**

Reference Manual

1 Introduction to Database Concepts

This reference manual describes in detail the structure and methods of the internal databases used within Marine. It is written for System Administrators who may be involved in maintaining projects, and the databases from which they are created.

1.1 Project

1.1.1 Project Organisation

In order to create data, a user must first create a Project.

A Project consists of:

- One each of System, Comms, and Misc DBs
- Multiple Design, Catalogue, Drawing, and Dictionary DBs
- Various picture files

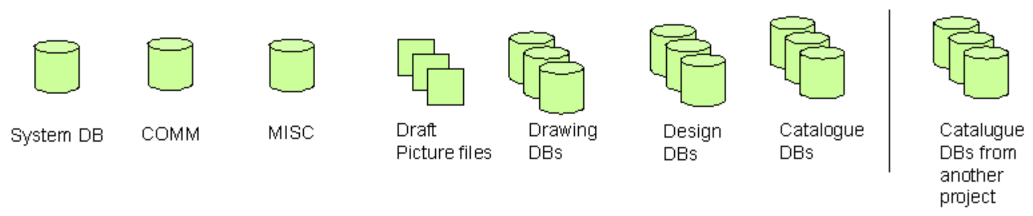
A project starts with just the System, Comms and Misc DBs.

The user will then have to create other DBs for users to work on. The various visible DB types are:

System	Contains details on DBs, MDBs, teams etc in the project
Dictionary	Contains UDA and UDET definitions
Property	Contains units for different properties
Catalogue	Contains catalogue and specification information
DESIGN	Contains Marine design information
Outfitting Draft	Contains drawing information
SPOOLER	Contains spool information
Materials	Contains hull material information
Diagrams	Contains Schematic information
Transaction	Used by Global to record transactions
SCHEMATIC	Contains PI&D (Schematic) information
MANU	Contains detailed manufacturing data
NSEQ	Stores name sequences

Typically there will be many DBs of each type within a project.

An example of a simple project is as follows:



DBs may be included from other projects.

Each DB has a unique DB number. User DBs have numbers in the range 1-7000. The range 7000-8000 is reserved for AVEVA supplied databases. The range 8000-8192 is reserved for system databases.

The DB number is assigned to the DB on creation. The user may specify the DB number. If not specified by the user then the next available DB number is used.

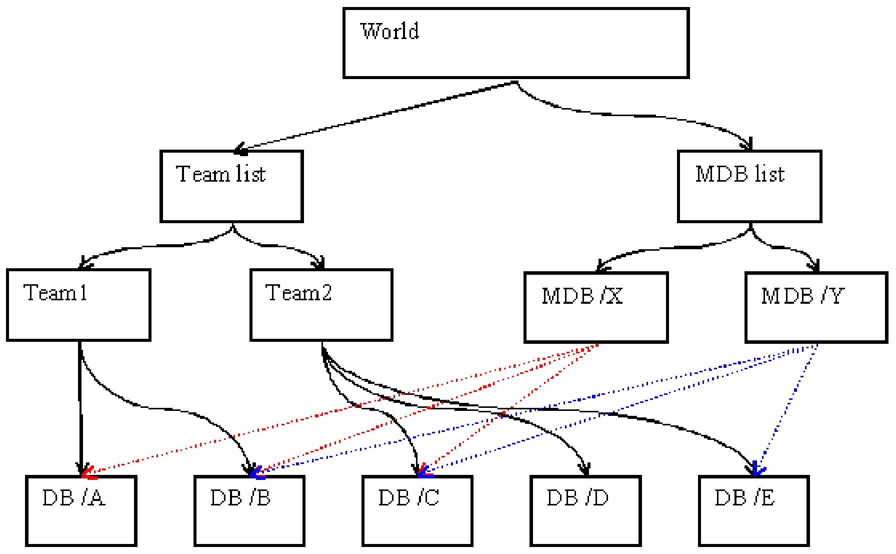
There may never be two DBs with the same DB number in a single project. Thus to include a DB from another project, that DB number must be unused in the target project.

1.1.2 Teams and MDBs

For ease of working DBs are grouped into Teams and MDBs (Multiple Databases). A DB belongs to one team, but may be in many MDBs. Users specify the MDB to work on when entering Outfitting.

Details of DBs, teams, and MDBs are stored in the system database.

An example is shown below.



It can be seen that DB /A is only in MDB /X, whereas DB /B is in both MDB /X and /Y.

Team access controls fundamental write access. Members of Team1 will always have write access to DBs /A and /B, and read access to the remainder. For members of Team2 it will be the opposite way around.

If a DB is included from another project then it is always read only regardless of team membership.

These concepts are discussed in detail in the *Administrator User Guide*.

1.1.3 Splitting Data Across Multiple DBs

Theoretically there need only be one DB of each DB type. The main reasons for there being more are:

- DBs are used as a fundamental means of access control. DBs are allocated to Teams, and only team members can modify a DB.
- Whilst the multiwrite facilities allow many writers per DB, it eases contention if the writers are not all accessing the same DB.
- The easiest way to run a Global project is to have different DBs writable at different locations.
- The granularity of propagation and distribution in Global is per DB
- It allows different permutations of MDBs.
- It allows specific DBs to be hidden from sub contractors
- Inclusion in other projects is done on a DB basis.

1.2 Database Elements

1.2.1 Introduction

All data in a Dabacon database is stored in elements. Every element has a type, e.g. BOX. The type of element determines the attributes available on the element.

Each DB type allows a different set of element types.

Database attributes are described in Database Attributes

The elements are organised in a primary hierarchy. This is described in Database hierarchy.

1.2.2 Reference Number

Every element has a reference number, which is assigned when the element is created, and is unique to that element within the project. The reference number comprises two 32 bit integers. This is displayed in the form:

=1234/6789

The first integer is composed from the database number and a bucket number. The bucket number allows for multiple users to access a database simultaneously without the risk of generating the same reference number; bucket numbers are allocated to users on a temporary bases starting at 1. In single write databases, the bucket number is always 1.

The second integer is a sequence number, starting at 0, and incrementing each time an element is created within that database and bucket.

The algorithm for allocating a reference number is:

1st part - DB number plus (bucket number * 8192)
2nd part - Increment starting from 0.

Thus, for example, for DB 1, the first element created will have a reference number of =8193/0 (this will be the world element since this is always created first).

The reference number is never changed once an element has been created.

1.2.3 Name

In Outfitting any element may be given a name. The name always starts with a '/'. At the user level, it is this name that is typically used to identify an element. Names may of course be changed, thus there is no guarantee that the element named '/FRED' today is the same as the element named '/FRED' yesterday. Names must be unique within a project.

An element need not have a name. For these elements Outfitting creates a constructed name, consisting of the relative position in the hierarchy up to the first named element.

e.g. BOX 2 OF SUBEQUIPMENT 1 OF /VESS1

Whilst the constructed name can be used to identify elements, its use is even more volatile than named elements, since the order of an element in a member's list may change.

1.2.4 Current Element

At the user level there is a concept of current element.

Most Outfitting commands act on the current element. This is often referred to as the CE. There is an extensive set of commands to navigate around the database changing the CE.

1.2.5 Changing Element Types

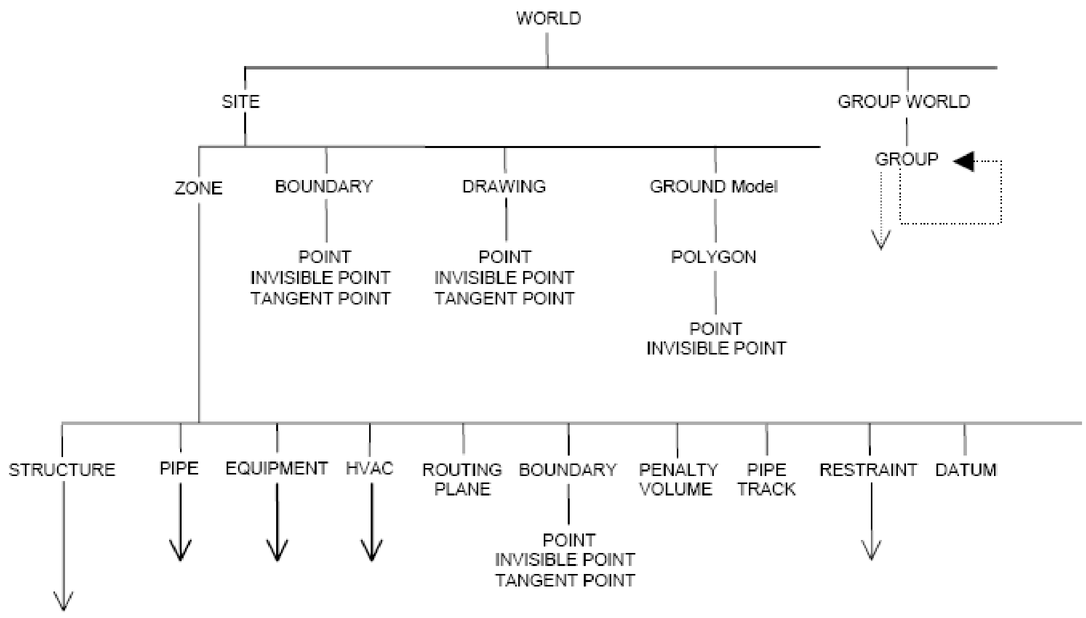
For most elements, the element type may never be changed after creation. For example, once created, an element of type SITE will always be of type SITE until deleted.

There are a few exceptions to this rule where it makes sense. For example, BENDs may be changed to ELBOs and vice versa.

1.3 Primary Database Hierarchy

1.3.1 Hierarchical Data Model

Each database consists of a hierarchy of elements. The allowed hierarchy is defined in a Database Schema. The database schema is the 'meta data', i.e. it is data about data. Database Schemas cannot be modified by users. The Database Schema for each database type is listed in the data model reference manual. An example of part of the Database Schema for DESIGN databases is shown below:



This schema shows which elements are allowed where. For example, a WORLD may own a SITE, or GROUPWORLD, whereas a SITE may own a ZONE, BOUNDARY, DRAWING or GROUND model.

The same element type may occur in more than one place in the schema. In the above example it can be seen that a BOUNDARY element may occur below a SITE or a ZONE.

All database schemas have a WORLD element at the root.

1.3.2 User Defined Hierarchies

The database schemas in Outfitting are fixed by AVEVA.

Users may, however, customise the allowed hierarchy using UDETS. (User Defined Element Types).

A UDET must be based on an existing system type. For example, a user may define a UDET :PUMP which is based on an EQUIPMENT. By default, a UDET will have the same allowed members and allowed owners as a base type. This can be customised to be a subset of that allowed on the base type, e.g. you might decide that SUBE are not allowed under a :PUMP even though they are allowed under an EQUI.

UDETs based on zones may own other UDETs based on zones. This allows very flexible data models to be built.

1.3.3 Element Instances

A new database starts with a single world element with name '/*'. Users will then create instances of other element types. For example, a system user might create an initial hierarchy of sites and zones in a DESIGN DB, leaving it to other users to create the actual ship items.

An element instance will always be created within the primary hierarchy. For example, a new ZONE element must be created under an existing SITE. It cannot be created as a 'freestanding' element outside the existing hierarchy.

The actual element hierarchy can be viewed with the Outfitting explorer.

All element instances within an MDB are accessible at all times.

Give an example here showing an explorer form plus explanation of what it is showing

1.3.4 Where the Primary Hierarchy is Used

The primary hierarchy is used as follows:

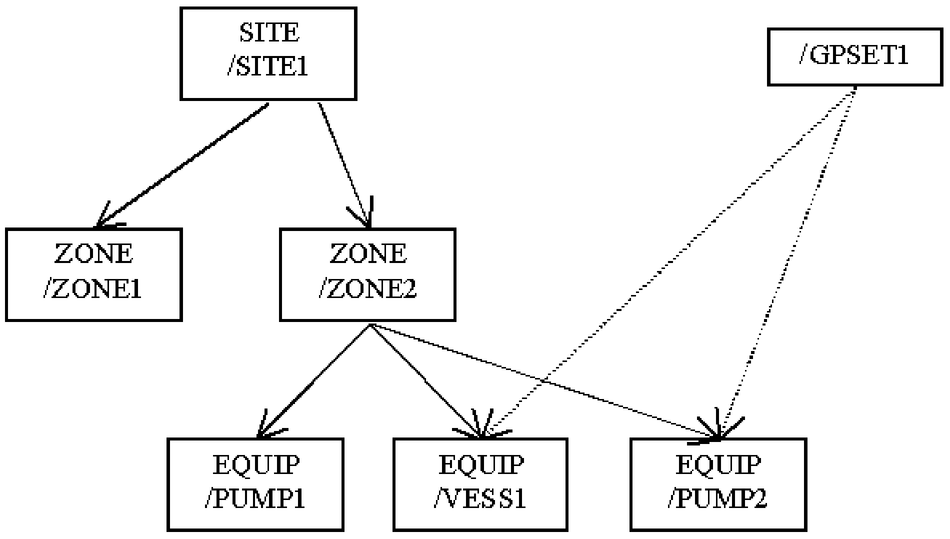
- It is used to create the 'constructed' name for unnamed elements.
- When an element is deleted, all descendants in the primary hierarchy are deleted.
- The COPY command will copy an element and all its primary descendants.
- Claiming elements relies on the primary hierarchy.
- Outfitting collections work on the primary hierarchy.
- Most commands assume that the action is to be performed on the given element and its descendants in its primary hierarchy, e.g. adding a ZONE to a 3D view will add everything below that ZONE.
- In the DESIGN DB, world positions and orientations are concatenated according to the primary hierarchy.

1.4 Secondary Hierarchies

1.4.1 Introduction

An element can only exist once in the primary data hierarchy. Secondary hierarchies, such as GPSETs, allow elements to appear more than once in the overall hierarchy. For example a PIPE will appear below a ZONE in the primary hierarchy. The same PIPE may also be added to a GPSET element. This is useful for collecting elements according to different criteria.

The diagram below shows a typical multi hierarchy where the secondary links are dotted.



Most commands will work on secondary hierarchies. For example, if /GPSET1 is added to a 3D view then this is equivalent to adding both /VESS1 and /PUMP2 to the 3D view.

However, there are exceptions to this. In particular deleting a GROUP will not delete the GROUP members; thus deleting /GPSET1 will not delete /VESS1 and /PUMP2

Unlike the Primary hierarchy, secondary hierarchies may span different DBs.

1.5 Database Attributes

Every element may have a number of attributes. All elements have the following attributes:

NAME	the element's name
TYPE	the element's type
LOCK	if set, then the element may not be modified
OWNER	the element's owner
MEMBERS	the current members of the element

The remaining attributes vary depending on the element type. The Database Schema defines which attributes are available on an element type. The allowed attributes for an element type may be ascertained using PML objects and other command queries.

Attributes may be one of the following types:

Integer	Ref
Integer Array	Ref Array
Real	Position
Real Array	Direction
Bool (or Logical)	Orientation
Bool (or Logical) Array	Attribute
String (or Text)	ElementType (or Noun)

A 'Ref' type is a pointer to another element. For example, on a BRANCH element the CREF attribute points to the connected NOZZLE. The use of Ref attributes enables Outfitting to model networks and other cross relationships.

The attribute type dictates how the attribute can be set with PML or specific syntax.

1.5.1 User Defined Attributes

Users can extend the allowed attributes for any element type, including a UDET, by defining UDAs (user defined attributes). For example, a user could define a UDA called :SUPPLIER of type string on all piping components. The newly defined UDA will then exist on all applicable elements, existing and new. If the definition of a UDA is changed then this will also apply to all existing instances.

Having defined a UDA, it is accessed in the same way as any other attribute.

1.5.2 Pseudo Attributes

In addition to the attributes stored on the database, there are a large number of pseudo attributes. The value of pseudo attributes is calculated at the time of making the query.

For example, the CUTLENGTH attribute on SCTN elements is a pseudo attribute calculated at the point of doing the query.

There is a lot of functionality presented via pseudo attributes. Currently there are over 1200 pseudo attributes.

Since the value of a pseudo attribute is calculated from other attributes, it is not generally possible to set their value directly.

1.5.3 Global Namespace for Attribute and Element Type Names

Attributes and element types have a global name space. This means that an attribute such as XLEN will have an identical meaning wherever it exists.

Similarly if an element type is valid in more than one database type, the definition of the element type will be identical in each.

1.6 Database Expressions and Rules

1.6.1 Expressions

Database expressions can be of the following types:

- algebraic
- boolean (or logical)
- text
- Element ID
- position
- direction
- orientation

The contents of an expression may contain the standard operator and mathematical functions along with the names of attributes and element identification.

Examples of expressions are:

Real Expression: (XLEN * YLEN * ZLEN * 2)

This expression simply multiplies the three attributes XLEN, YLEN, ZLEN together and then multiplies by two.

The attributes refer to the current element. If attributes of other elements are required then the OF syntax is used.

Boolean expression: (PURP EQ 'HS' AND AREA OF OWNER EQ 1)

The 'OF' keyword ensures that the AREA attribute is obtained from the owner of the current element rather than the current element itself.

The main uses of expressions are:

- Catalogue parameterisation
- Template parameterisation
- Rules
- Drafting line styles
- User level collections and report

Database expressions are very similar to PML expressions. The major difference is that database expressions may not contain other PML variables or functions. E.g. (XLEN * !MYVAR) is not a valid database expression.

1.6.2 Rules

An attribute may be defined as a rule. For example, the attribute XLEN may be defined as a rule by the expression (YLEN * 2).

The OF syntax is often used in Rule expressions to refer to other elements, e.g. (YLEN OF / FRED * 2)

The result of the rule is stored against the attribute as for other attributes.

There are commands to recalculate the rule.

Rules may be either static or dynamic. If static, then the rule result will only be recalculated on demand. If dynamic, then the result will be recalculated every time an attribute within the expression changes, E.g. for the above rule, if YLEN is modified, then XLEN will be recalculated automatically. The dynamic linkage of rules may be across elements and across DBs.

1.7 Dumping out the Database

1.7.1 Data Listings

Data listings (DATALs) capture the state of the database in the form of Outfitting commands. All element data including all attributes, UDAs and rules will be captured. They are similar in concept to XML output. These files can then be read back in via the command line.

Data listings are used as follows:

- Long term archiving
- Copying parts of a DB between projects
- For general information.

1.7.2 Reconfigurer

Reconfigurer is similar to Datal in that it dumps out the state of the data.

The data can be dumped to either binary or text file. Using binary files is quickest.

Reconfigurer is faster than Datal and is recommended if whole DBs or world level elements are to be transferred. Datal or the copy facilities is recommended if lower level elements are to be transferred.

1.8 Database Modifications

1.8.1 Overview

The fundamental modifications allowed are:

Element creation

Element deletion

Element copy

Element move

Attribute modification

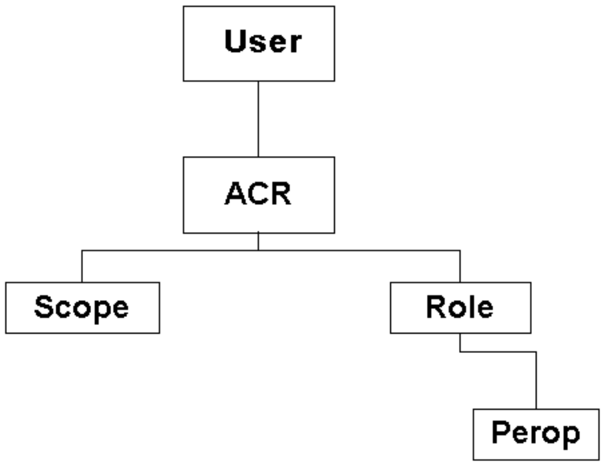
1.9 Data Access Control (DACs)

Data Access Control (DAC) is the mechanism that protects information handled by the system from accidental or unauthorised manipulation.

For a more detailed description of the basic functionality and administration of DAC refer to the *Administrator User Guide*.

The basic access control available is known as 'Team Owning Databases'. It implements access control on database level by simply giving the members of the team owning the database full access and others read-only to data held in particular databases.

A more sophisticated access control is implemented in the form of Access Control Rights (ACRs). ACR allows the administrator of the system to apply a more fine grained access control over the model. The following figure illustrates the DAC database hierarchy.



An ACR is defined through two entities:

- A ROLE, which is a collection of rules called Permissible Operations (PEROPs).
- A SCOPE, which defines to what part of the model the ROLE applies. The SCOPE may be an expression, E.g. all ZONE WHERE (FUNC eq 'TEAMA')

A PEROP defines the access rights given for a number of pre-defined operations for one or more elements.

One or more ACRs may be assigned to a user granting and denying access to the model.

For a user to gain update access to a particular element two rules apply:

- At least one PEROP in a ROLE assigned to a USER must grant the update operation.
- No one PEROP must explicitly deny the operation.

Management tools are available for DAC through the ADMIN module. Control of DAC is also available through PML.

A PEROP consists of three parts:

- The Element it applies to
- The operations which can be performed on those elements
- Optionally the Attributes that may be modified.

The PEROP may further restrict the elements it applies to by a qualifying condition. The qualifying conditions is an Outfitting statement that should evaluate to true to qualify the PEROP.

The following operations are available through PEROPs

Create
Modify
Delete
Claim
Issue
Drop
Output
Export
Copy

Each of these operations may be set to

Allow	The operation is permitted
Disallow	The operation is not permitted
Ignore	The PEROP does not define whether this operation is permitted or not

Optionally the PEROP may further restrict which attributes it allows modification to by specifying a list of attributes that it either includes or excludes from allowing modification to.

The PEROP also holds the message that the system will issue if the PEROP denies attempted operation.

1.9.1 Errors Applicable to all Modifications

The following checks are applied to all modifications:

- Check access control
- Check that the DB is open in write
- Check that the element's LOCK flag is false
- If a multiwrite DB then do a claim check, and claim if needed

The claiming process is described in Claiming Elements

1.9.2 Integrity of Modifications

The engineering integrity is always maintained for any database modification.

The integrity checks are applied below the database interface. Thus modifying the database is always safe whether done via PML commands or C#.

The checks are applied to individual attributes and element types. For example the OBST attribute can only ever be set to 0,1 or 2. Outfitting will always check that this is the case prior to allowing the modification.

1.9.3 Element Creation

Elements may be created. They are always created one at a time, and may only be created at a legitimate point in the primary hierarchy.

On creation, a unique reference number will be assigned. The method by which the default reference number is generated is described in *User Defined Hierarchies*.

It is possible to create an element with a specified reference number, provided it is unused and valid for the DB. This functionality is provided for certain specialised situations (such as recreating an exact copy of a DB, so that all references to elements from other DBs remain valid), and is not recommended for general use.

The attributes will be set to their default values. In some cases the default attribute values are cascaded down from the owning element.

1.9.4 Element Deletion

Elements may be deleted. All elements below the deleted element in the primary hierarchy will also be deleted.

Reference numbers of deleted elements are never reused.

1.9.5 Element Copy

Elements may be copied. There are options to copy a single element or an element and all its descendents. Elements may be copied between DBs. Any cross references entirely within the copied structure will be updated to point to the newly created elements.

Elements are always copied on top of an existing element of the same type.

There are various options on the copy command to allow:

- The copied elements to be renamed according to a given criteria
- Whether any attribute rules are to be rerun on the copied element. (Rules are described in *Reconfigurer*)

Additional potential errors at create are:

- The element may not be copied to an element of a different type

1.9.6 Element Move

Elements may be moved to a different point in the DB or to a different DB.

The Element and all its descendants will be moved.

If the element is moved to a different DB, then its reference number is changed. All reference attributes pointing to the moved structure will be updated.

Additional potential errors at move are:

- The element is not allowed in the members list of the new owner

1.9.7 Attribute Modification

The following checks are applied when modifying attributes:

1. the attribute value is the right type
2. the attribute is valid for the given element

Modifying Related Attributes

Sometimes modifying one attribute will actually cause a number of attributes to change. There are two main cases where this might happen:

1. There is a dynamic rule on another element dependent on this element (see section Rules for description of rules)
2. There is built in code that keeps a number of attributes synchronised. This is used for some Draft attributes and some cross references.

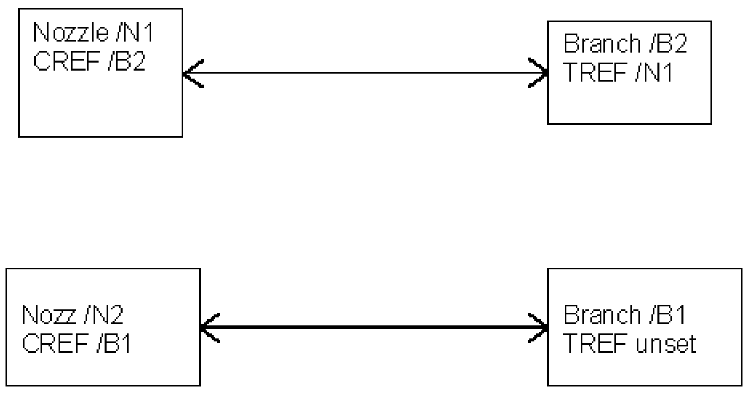
Changing Cross Referenced Attributes

The integrity of cross referenced attributes is maintained when one end of the connection is changed. Changing one end of a connection will also change the following:

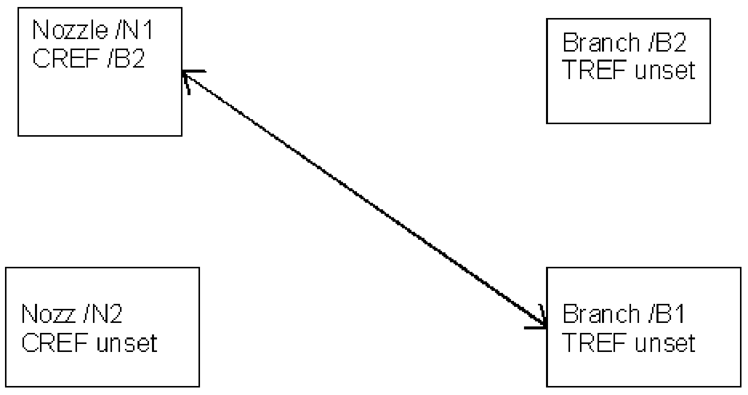
1. If there is an existing connection, the corresponding attribute on the element at the other end of the existing connection, is unset.
2. For the new element connected, if this is already connected, then this connection is unset, which itself may change the element at the other end.

In essence, changing one value may result in four elements being updated.

For example, consider the following:



After setting the CREF on /N1 to /B1, the end result is:



1.9.8 Effect of Modifications on Dynamic Rules

A dynamic rule will automatically respond to changes which affect the attributes referred to in the rule.

For example, we set a rule on YLEN of /MYBOX to be (YLEN OF /FRED * 2). Thus if YLEN on /FRED changes then YLEN on /MYBOX will be updated automatically. However there are reasons why the automatic propagation of dynamic rules may fail, as follows:

1. The elements are in different DBs and the DB containing /MYBOX is not in the MDB
2. The elements are in different DBs and the DB containing /MYBOX is read only

3. It is not possible to claim /MYBOX
4. /MYBOX is locked
5. There is a DAC preventing the change

Note also that only static rules are not automatically updated. For these reasons there are commands to verify that rule results are up to date.

1.10 Database Sessions

1.10.1 Savework/Getwork

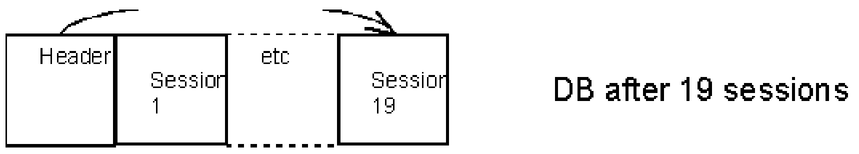
Data is only saved to the database on demand. Similarly a user will only see changes made by others on demand. In order to make changes visible to other users two steps must occur:

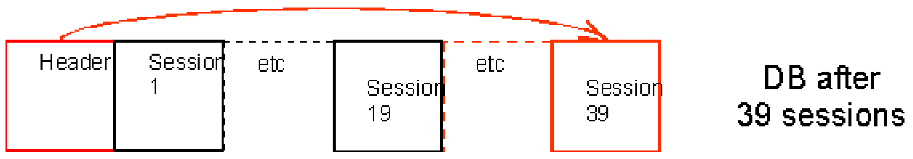
1. The user making the changes must do a Savework
2. The reader must do a Getwork

For most applications, the savework/getwork actions are totally in the hands of the user.

1.10.2 Sessions

When a savework is made a new session will be created on the database. The changed data will always be written to the end of the file. This represents the 'delta' from the previous session. Details such as date, user, session description are stored as part of the session data. There is always a pointer from the database header to the last session on the database.





Internally there is a linked list between sessions. It is worth reiterating that once a session is written, it will never be changed. Thus if a user is looking at session 19, then his view of the data will never change in any way regardless of any sessions created by other users. If the user does want to see the changes made by others then he must do a 'Getwork'. 'Getwork' will always reposition the user to view the latest session. Thus in the above example if a user originally looking at session 19 did a Getwork then he would now be looking at session 39.

1.10.3 Session History

Overview

The Database will preserve the full session history. Thus at any point it is possible to find out what was changed when and by whom. The system can report on changes down to the attribute level.

The list of facilities include:

- Comparing elements to an old session
- Dataling out changes since a given session
- Setting a comparison session
- Creating a stamp
- Various pseudo attributes

Comparing Elements to an Old Session

The DIFF command can be used to report on changes. For example, if the user were to modify a couple of attributes on an equipment, and add a new primitive, then the DIFF command could be used to report on the changes. The output from the command might be:

```
Local comparison for Database items:-
 /VESS1
/VESS1 [=15752/201] has been modified
Member list has changed
  List member /EXTRACYL created
Description has changed
    Old value= my description
    New value= my new description
Area has changed
    Old value= 999
    New value= 100
/EXTRACYL [=15752/1326] has been created
2 changed elements found
```

By default, the DIFF command will report the changes made by the user in the current working session, that is to say, since the last savework. It is also possible to specify a given session number, a date and time, or a stamp (see *Creating a Stamp*) in order to see the differences since then.

Dataling out Changes Since a Given Session

The OUTPUT command may be used to record changes since a given session. The Datal file will then contain the commands that reproduce the updates made since the given session. The file can then be read back in to reproduce the changes. This is convenient where bits of data have been copied between projects, and the copied data needs to be updated with changes made to the original.

Reverse changes can also be output. The Datal file will then contain the commands that remove the updates made since the given session. This is a convenient way for restoring part of a model back to how it was at an earlier point.

1.10.4 Creating a Stamp

It is often convenient to mark a set of DBs at particular milestones in the project. The 'Stamp' functionality allows this. It is then straight forward to find out what has changed since the stamp, or to view the data as it was at that time.

Stamps can only be created within ADMIN.

1.10.5 Setting a Comparison Date

Within Outfitting a comparison date can be set. A convenient way of doing this is to use a stamp. However the comparison date may also be set to an explicit session on a particular extract. If the comparison session is on a different extract then the extract must be an ascendant of the current extract.

Any query may then be done at the old session using the 'OLD' keyword. e.g.

```
Q OLD XLEN
```

This would return the value of XLEN at the comparison session.

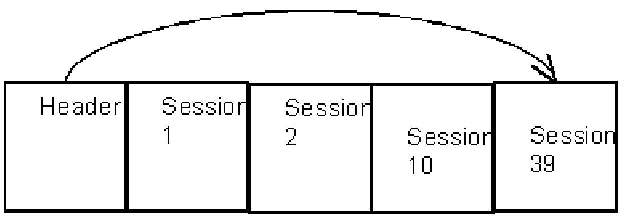
Pseudo Attributes for Comparisons

There are a number of pseudo attributes that can be used to do comparisons. These are listed in the database reference manual.

1.10.6 Merging Changes

As a result of storing all changes, Outfitting databases will grow relatively quickly. The user may compress a DB to reduce its size by merging multiple sessions together using the MERGE CHANGES command in ADMIN. The user may compress all sessions, or a range of sessions.

Any sessions used in stamps will always be preserved. Thus before compressing a DB the user should create stamps to preserve any comparison points that might be needed. Sessions 1,2 and the last session are also always preserved. Thus for the previous example, if the user decides to do a MERGE CHANGES on a database having set a stamp on session 10, the resultant DB will look like:



It can be seen that as well as sessions 10, sessions 1,2 and 39 are kept. The changes in session 10 now hold the accumulated changes for sessions 3-10, and Session 39 actually holds the accumulated changes for sessions 11 to 39.

The MERGE CHANGES command is discussed in the ADMIN manual.

1.11 Multiwrite Working

Dabacon DBs may be either 'UPDATE' or 'MULTIWRITE.

UPDATE DBs allow only one user to have write access at any given time, although multiple users may still have simultaneous read access.

MULTIWRITE DBs allow multiple simultaneous users with write and read access.

1.11.1 Multiwrite Strategy

The Outfitting Database employs two techniques to allow multiple writers.

A claiming mechanism - User must claim an element at the point of making a modification. This will lock the element preventing other users making modifications.

A Last Back Wins strategy- For some changes a 'last back win' strategy is used rather than claim locks. With this strategy two users may change the same element. Any changes are merged back in. The merging is done at the attribute level. If two users change the same attribute then the last save wins. Places where this strategy is used are:

- Some connection attributes. e.g changing a TREF attribute on a branch does not require the BRANCH to be claimed.
- Member lists containing primary elements are always merged back. e.g. creating a ZONE below a SITE doe NOT require the SITE to be claimed,
- Changes issued from variant extracts are always merged back in.
- Dynamic rule linkage
- Spatial map values

1.11.2 Claiming Elements

The level of claiming is at the 'primary' element level. Examples of primary element types are SITE, ZONE, EQUI, SUBE. Examples of non primary elements are primitives such as BOX. When you need to modify a non primary element then the first primary element up the hierarchy must be claimed out. E.g. to modify a BOX, then the owning EQUI or SUBE must be claimed.

When working on multiwrite DBs, users may either explicitly claim elements to work on, or let the system implicitly claim elements for them. The implicit claim will occur at the point of making a modification.

There are a number of reasons why an element may not be claimed:

1. The element is claimed by another user
2. The element is claimed by an extract
3. The element has been changed since this user last did a GETWORK or SAVEWORK. To remedy this, the user must do a GETWORK first.

If a list of elements is claimed, and some in the list fail, then the remaining elements will still be claimed.

1.11.3 Releasing Elements

Having claimed an element, a user may release it, thus allowing others to change it.

An element may not be released if changes are outstanding. The user must do a SAVEWORK first.

On leaving a module all elements will be released for that user.

If a list of elements is released, and some in the list fail, then the remaining elements will still be released.

1.11.4 Performance Considerations

Every time a claim/release is made the underlying DB is accessed. To minimise such access, as many elements as possible should be done in one go.

1.11.5 Potential Conflicts at SAVEWORK/GETWORK in a Multiwrite Environment

There are a number of potential problems which are not discovered until SAVEWORK or GETWORK.

1. Two users could simultaneously use the same name for an element. This may not be discovered until the second user attempts to do a SAVEWORK or a GETWORK. In this case the second user must rename the offending element before SAVEWORK or GETWORK is allowed.
2. One user may insert a primary element in another element's list, say /PIPE1, whilst a second user deletes /PIPE1. Again SAVEWORK or GETWORK will throw an error, and the user will have to move or delete the offending branch in order for SAVEWORK or GETWORK to take place.
3. The opposite of (ii) can also occur. i.e. you have deleted a primary element in which another user has created an item. In this case the user must QUIT the changes.

If any error occurs at SAVEWORK or GETWORK then the entire operation is aborted.

1.12 Extracts

1.12.1 Extracts

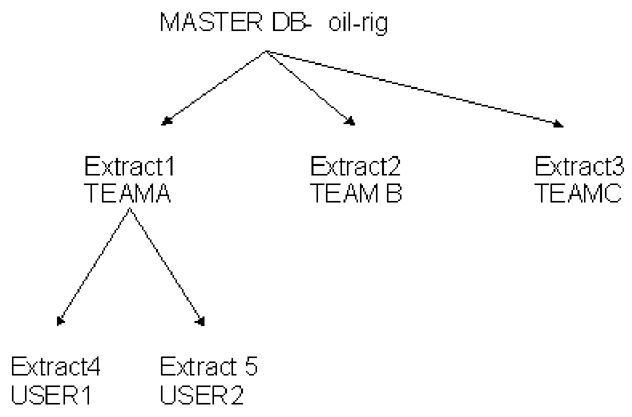
Extracts are an extension of the multiwrite facilities.

The extra functionality offered by extracts is:

- They allow long term claims, i.e. Elements are not released on module switch.
- The issuing of data is separated from SAVEWORK.
- A partial set of changes may be issued, rather than the whole lot.
- A partial set of changes may be dropped, hence losing the changes.
- Allows variants to be tried and maintained.
- Allows a 'last back wins' when issuing from variants
- Users may have their own private work space.
- Users can use extracts to implement an approval/work cycle, i.e. the issuing of data from one extract to another could correspond to given criteria being met. Other users could then read the approved data rather than the working data.

1.12.2 Creating Extracts

Extracts are created from existing multiwrite DBs. The existing DB then becomes the Master DB. Many extracts may be created off the same Master DBs. Extracts may also be created from other extracts. The term extract family is used to refer to all extracts created directly or indirectly off a Master DB. Example of an extract family:



In this extract family, three extracts were created below the Master DB. Two further extracts were then created below Extract1.

There may be up to 8000 extracts in an extract family.

Extracts may be included in an MDB as for any other DB. Two extracts from the same extract family cannot be included in the same MDB.

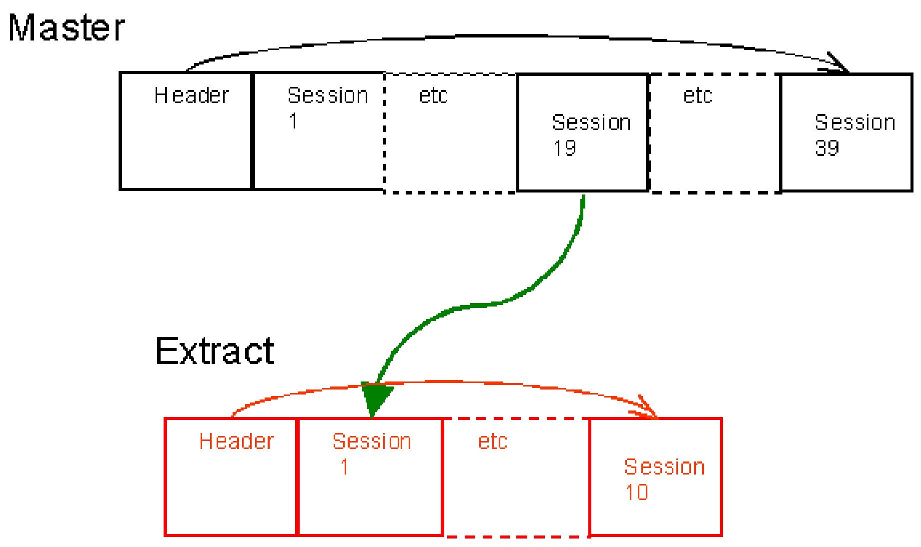
1.12.3 Restrictions on Extracts

It is not be possible to:

- COPY an extract
- INCLUDE an extract from a foreign project without its parent extract being included first.
- EXCLUDE an extract/DB unless all child extracts have been excluded.

1.12.4 Extract Sessions

An extract will have its own set of sessions. This is illustrated below:



In this example the extract DB was created when the Master DB had 19 sessions. The extract thus represents a branching of the model from session 19. Changes were then made to the Master and to the Extract. The Extract has had nine more sessions created (sessions 2-10). The Master has had 20 more sessions added (sessions 20 - 39).

Changes made to an extract can be flushed back to the Master DB. Similarly the extract may be refreshed with changes made to the Master.

1.12.5 MERGE CHANGES on Extracts

When a 'MERGE CHANGE' operation is performed on a DB with extracts, all the lower extracts have to be changed to take account of this. Thus doing a 'MERGE CHANGE' on a DB with extracts should not be undertaken lightly. The opposite is not needed, i.e. MERGE CHANGES on a child does not require the parent extract to be merged.

The following restrictions apply to MERGE CHANGES:

1. Any sessions 'linked' to child extracts are preserved.
2. There may be no users on any lower extracts.

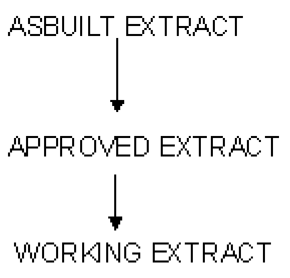
To minimise the sessions preserved in (1) it is suggested that a bottom up approach is followed when doing 'MERGE CHANGES'.

1.12.6 Extract Claims/Releases

In order to modify an element in an extract, the element must be claimed to the extract. The principals of extract claiming are exactly the same as standard claiming, i.e, the granularity of extract claims is at the level of primary elements.

Extract claims will work up through the extract levels, extract claiming as necessary, i.e, the user need not do a separate claim for each level of extract.

For example, consider a three level extract as follows:



If a user does an extract claim to the Working Extract the following logic will be used:

```
Is element claimed out to WORKING already?
        -if YES
              do nothing
        -if NO
              Is element claimed to APPROVED extract?
```

-if NO

Claim from ASBUILT to APPROVED

Then claim from APPROVED to WORKING

-if YES-

Claim from APPROVED to WORKING

The extract claim may fail for the same reasons that a user claim may fail, i.e.:

- Another user/extract has the item claimed
- The element is modified in a later version, hence a refresh is needed first.

Unlike user claims, extract claims stay with the extract across SAVEWORKs.

If a list of elements is extract claimed, and some in the list fail, then the remaining elements will still be extract claimed.

1.12.7 Extract Release

Extract claims may be released in the same way as user claims.

An extract release will not be permitted if:

1. Updates are outstanding on that extract
2. The element is currently claimed out to a user or to a lower level extract

If a list of elements is extract released, and some in the list fail, then the remaining elements will still be extract released.

1.12.8 User Claims/Releases on an Extract

An extract is itself a multiuser DB, thus more than one user may work on an extract. Standard user claims and releases are thus also applicable to extracts.

1.12.9 Variants

Variants are like standard extracts except that there is no extract claiming of elements between the variant and its parent extract. Any elements may thus be modified. This has the advantage that many users can potentially change the same element at the same time in a different variant. The disadvantage is that conflicts are not discovered until the time of flush.

There are no restrictions on where variants are located in the extract tree, e.g. variants may own other normal extracts or other variant extracts. If a variants owns standard extracts, then the variant acts as a root for subsequent extract claims.

1.12.10 Extract Operations

The following operations are allowed:

Drop	This is the process of losing modifications done locally on an extract combined with the transfer of write extract back to the parent extract.
Partial Drop	This is the process of losing modifications done locally on a subset of elements, combined with the transfer of write extract back to the parent extract.

Issue	The local changes are copied to the parent extract, and the elements are released.
Flush	This term is used for issuing without doing a release.
Partial Issue	The Issuing of a subset of the modifications made in the current extract to the parent extract.
Refresh	The mechanism by which an extract is updated with changes made in the parent DB.

The refresh functionality is needed since users work on a constant view of the parent extract DB. Thus they will not see other users' issues until they do a REFRESH. It is akin to the GETWORK functionality in a single multiwrite DB.

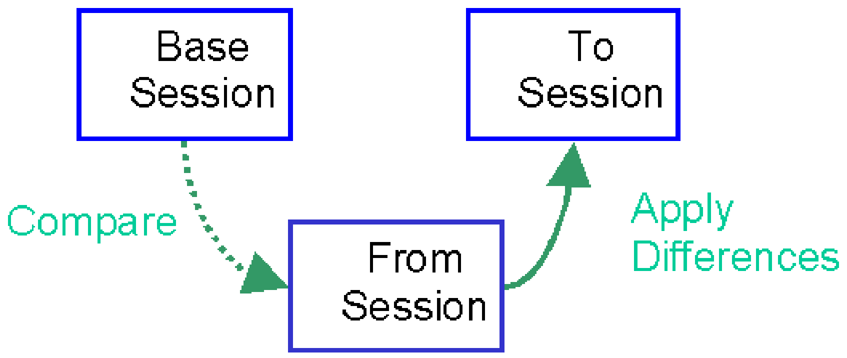
All flushing, issuing, releasing and dropping operations work on one level of extract only. A Refresh can be done all the way up the tree using just one command.

If an extract operation fails, then the entire operation is aborted.

1.12.11 Merge Algorithm

On issue or flush, changes made in an extract will be merged back to the parent extract.

The basic approach is that any changes made to the extract are applied to the parent extract, as shown below:



The granularity of this merge is at the attribute level, i.e. two users can change different attributes on the same element and merge their changes together. If they modify the same attribute then a 'last back win' strategy is used.

Outfitting ensures that all merges are valid at the raw database level, i.e. the data will be DICE (Database Integrity Checker) clean. However it is not possible to ensure that the data is consistent in engineering terms. Thus it is highly recommended that when variant data is flushed back, Datacon checks and Clasher checks are run on the resulting version.

The definition of the different sessions for issue and flush are:

Base Session	Session on parent when Refresh was last done
From Session	From session on child extract
To Session	New session on parent

The definition of the different sessions for refresh are:

Base Session	Session on parent when Refresh was last done
From Session	From session on child extract
To Session	New session on parent

The definition of the different sessions for drop are:

Base Session	Session on parent when Refresh was last done
From Session	From session on child extract
To Session	New session on child

Note: The standard flush and issue commands also do a refresh. This ensures that there is a suitable base session for the next extract operation.

The drop command compares the elements that are NOT to be dropped and applies the changes to create a new session.

The same algorithm is used for SAVEWORK and GETWORK.

There are two exceptions to the merge criteria as follows:

1. Once an element is deleted, then it stays deleted regardless of any conflicts in merging. For example, user 1 deletes /BOX, whereas user 2 changes an attribute of /BOX. If user 1 issues his change before user 2, then user 2's change will have no affect.
2. If the element type is changed, then the merge is done at the element level rather than the attribute level, i.e. all the current attribute values are taken regardless of whether they have changed. Any attribute changes made by other users will thus be lost.

1.12.12 Dealing with Deleted Elements

There are three ways of denoting deleted items for a partial operation.

1. The reference number of the deleted item can be specified. The reference number can be obtained by opening an MDB which includes the parent extract and navigating to that element, and then querying ('QUERY REFNO')
2. If the 'HIERARCHY' keyword is used, then any deleted items within that hierarchy will be included.
3. For primary elements, if the name of the deleted element has been reused, then flushing the element with the new name will flush the deleted element. For example, if the user deletes PIPE /PIPE1 and then recreates it, then flushing /PIPE1 will also flush the deleted pipe.

1.12.13 Flushing Connected Items

For two way connections, it is often desirable to flush both ends of the connection in order to preserve engineering consistency. There are addition options that allow the connected items to be flushed.

1.12.14 Errors for Extract Operations

Potential problems at issue and refresh are the same as for SAVEWORK and GETWORK on multiwrite DBs, i.e. there could be a name clash, or an owning element could be deleted. Such problems will need to resolved in the extract prior to issuing being allowed.

If doing a partial issue or flush or if issuing from an extract, then extra checks are applied as follows:

- Where a non primary element has changed owner, then the old primary owner and the new primary owner must both be issued back at the same time.
- If an element has been unnamed, and the name reused, then both elements must be flushed back together.
- If an element and its owner have been created then:
  1. if it is included in a partial flushback, then so must its owner.
  2. if the owner is include in a partial drop, then so must the element itself
- if an element and its owner have been deleted then:
  1. if it is included in a partial drop, then so must it's owner.
  2. if the owner is included in a partial flush, then so must it.

Where an element has not been claimed, then Drop can still be used to lose the local changes.

Note: When a partial issue or drop is made there is no guarantee that the data is 'Engineering correct'. Users are advised to run Clasher and/or Datacon on the resultant version.

1.12.15 Performance Considerations

A new session is created for every flush operation. Thus it is much better to flush a large number of elements in one go rather than flush them individually.

1.13 Global Working

1.13.1 Overview

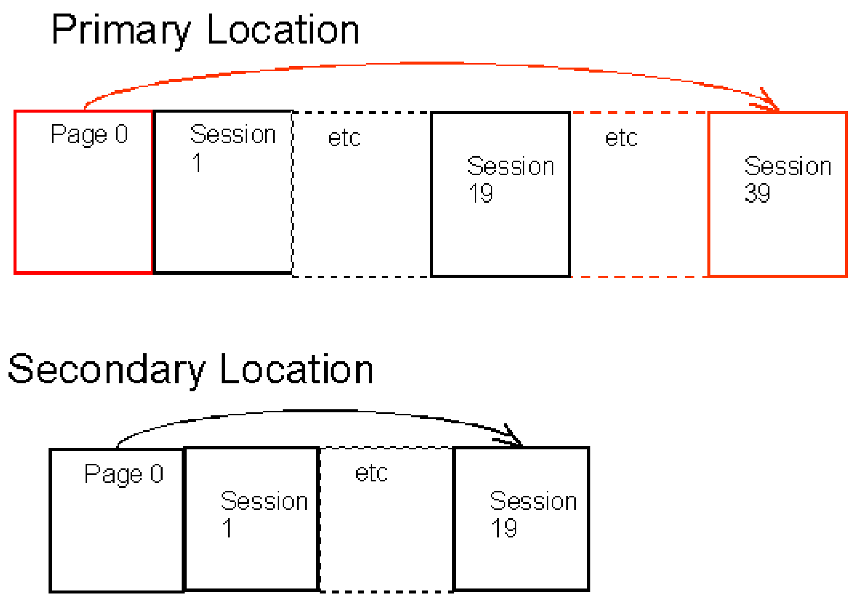
Global allows DBs to be spread across different locations. Global propagates changes from one location to another. The Global daemon does the propagation.

With global, there may be copies of a database or extract at multiple locations, but only one copy may be writeable. A database is said to be primary at a given location if it is writeable there, and secondary if it is not. A DB may be made primary at any location.

Different extracts from the same extract family may be primary at different locations. This allows multiple different locations to modify the same DB.

1.13.2 Global Propagation

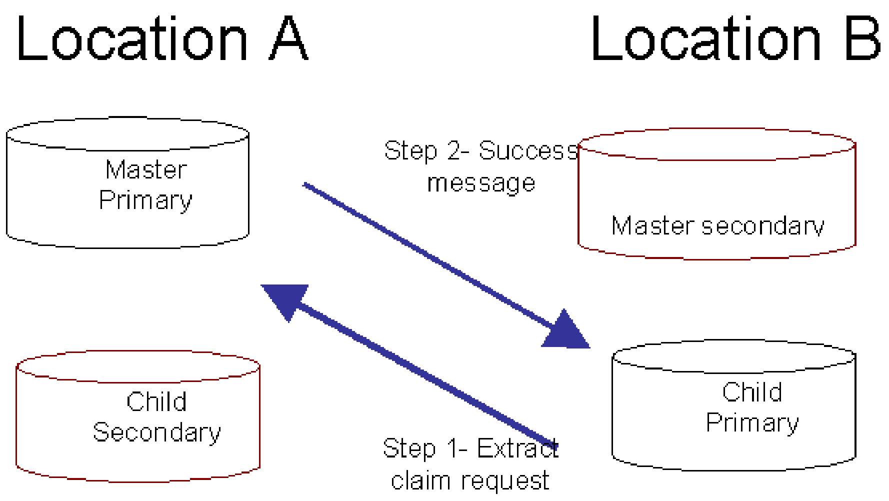
Changes made to a primary DB are propagated to the read only secondary DBs. The propagation algorithm just sends the new sessions. For example:



If this case the propagation algorithm will send the sessions 20 to 39 to the secondary location.

1.13.3 Extract Claiming/Releasing with Global

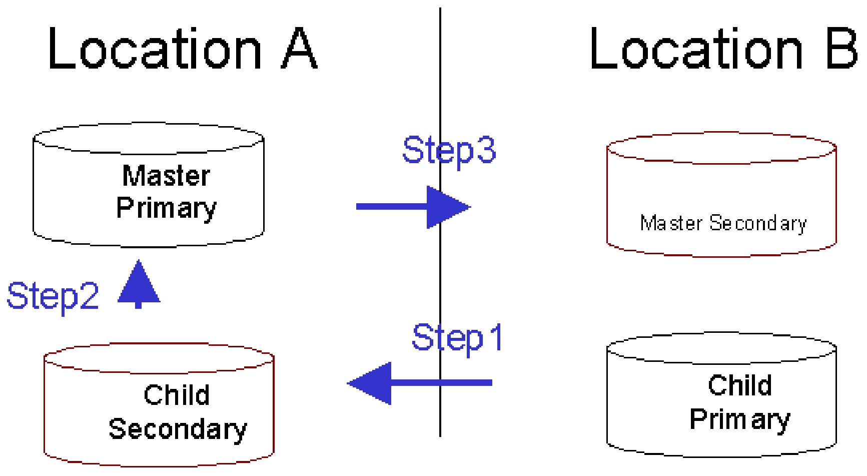
If the two extracts are primary at the same location, then the extract claim/release operations are the same as for non global projects. If the two extracts are primary at different locations then the claim/release goes via the daemon. For example:



The extract claim operation is thus asynchronous. The user has to wait to discover if the claim succeeded or not.

1.13.4 Flushing with Global

If the two extracts are primary at the same location, then the flush operation is the same as for non global projects. If the two extracts are primary at different locations then the steps are as follows:



The steps are:

1. Propagate sessions on the child from location B to location A
2. Flush the Master at location A with changes
3. Propagate the new sessions on the Master at location A to location B (at next update)

Step 2 could fail for normal reasons, e.g. a name clash. If so the primary child extract needs to be informed so that next time it uses the correct base session for comparison purposes. At the command level this is the 'EXTRACT FLUSH RESET' command.

1.13.5 Merge Changes for Global Extracts

In the Global environment MERGE CHANGES will only be allowed where all lower extracts are also primary on this location.

1.14 Undo Capabilities

1.14.1 Undo/Redo within a Session

The Outfitting database has a built in undo/redo capability. Applications may define a start/ end transaction and wind back to the start of that transaction.

Internally this is implemented using 'micro' sessions. Each micro session represents the start of each transaction.

An undo can not be done across a SAVEWORK.

1.14.2 Backtrack/Rewind

The system administrator may remove the last one or many sessions from the DB.

2 Database Navigation and Query Syntax

This section covers aspects of database navigation. Many examples are given during a DESIGN session but are also relevant to the Outfitting Draft module.

2.1 Current Element

The database has a concept of current element. This is often referred to as the CE. The current element is highlighted in the explorer. In the 3D view the current element is shown in a different colour.

Many PML commands work on the current element.

The current element can be changed in the following ways:

- By picking an element in the explorer
- By picking an element in the 3D view
- By typing an element name into the name box
- By typing a navigation command at the command line

2.2 Current Position

There is also a concept of a current position. The concept of current position is only used when creating elements or when navigating down to the next level.

By default the current position is before the first member.

This is described further in *Climb Up*.

2.3 PML DBRef Object

PML supports a DBREF object. A DBREF object identifies an element in the Outfitting database. There are various methods available on the DBREF object. The methods for a DBREF are described in the software customisation manual.

2.4 PML !!CE Variable

There is a global PML DBREF variable called !!CE that tracks the database current element. This object may be used/queried at any time.

2.5 Specifying the Standard Name

This is the simplest way of navigating around the database. Just enter the name of the element at the command line. A name is always preceded by a slash.

e.g.

/PUMP99

/BRANCH2

/BRANCH2 will now be the CE.

2.6 Specifying the Constructed Name

Unnamed elements always have a constructed name. The constructed name consists of:

- the type and relative position in it's owners list.
- the OF keyword
- the constructed name of it's parent

If the constructed name is given, that element will become the CE.

e.g.

BOX 2 OF /PUMP99

NBOX 1 of CYL 2 of EQUI 4 of ZONE 9 of /MYSITE

If the element is a UDET then the UDET name must be used instead of the system type.

e.g. NBOX 1 OF :MYCYL 1 OF :PUMP 3 of ZONE 9 of /MYSITE

2.7 Specifying the World

The following syntax accesses the world element by name, type:

/* or WORLD

2.8 Specifying the Refno

The reference number can always be used to navigate to an element.

=1234/5678

2.9 Specifying a Relative Position in the Hierarchy

Relative navigation can be done using a number of commands as follows:

- Climb up
- Move within current level
- Move to next lower level

2.9.1 Climb Up

The following syntax is valid:

OWNER	climb to owning element. The owning element becomes the CE. The *current position* is then before the first member
END	climbs to owning element. The owning element becomes the CE. The *current position* is at the previous element
<Element type>	climb to element of that type. This element becomes the new CE. This leaves the *current position* at the immediate member element that was climbed through.

e.g. consider the following hierarchy:

```
/*
/MYSITE
/MYZONE
/MYEQUI
/MYBOX
```

If the CE is /MYBOX, then:

OWNER	The CE becomes /MYEQUI. The current position is now before the first member.
END	Also climbs to /MYEQUI, but leaves the current position at /MYBOX
EQUI	Also climbs to /MYEQUI, and leaves the current position at /MYBOX
SITE	Climbs to /MYSITE, and leaves the current position at /MYZONE

2.9.2 Move within the Current Level

Next	Goes to next element in at current level
Next int	Goes to next nth element at current level
Next <element type>	Goes to next element of given type at current level
Next int <element type>	Goes to next nth element of given type at current level
Prev	Goes to prev element at current level
Prev int	Goes to prev nth element at current level
Prev <element type>	to previous element of given type at current level
Prev int<element type>	to previous nth element of given type at current level
First	Goes to first element at current level
First int	Goes to nth element at current level
First <element type>	Go to first element of given type
First int <element type>	Go to nth element of given type
Last	Go to last element at current level

Last int	Go to nth from last element at current level
Last <element type>	Go to last element of given type
Last int <element type>	Go to nth last element of given type
<element type> int	This is the same as ‘First int <element type>’

If a UDET, then the UDET type must be given.

Example

Current list is:

1 BOX /MyBoxA
2 CYL /MyCylA
3 CYL /MyCylB
4 RTOR /MyRtorA
5 CYL /MyCylC
6 BOX /MyBoxB
7 BOX /MyBoxC
8 CYL /MyCylD
9 BOX /MyBoxD

The Current element is /MyCylC, as highlighted.

NEXT	Moves CE to /MyBoxB
NEXT 3	Moves CE to /MyCylD
NEXT CYL	Moves CE to /MyCylD
NEXT 3 BOX	Moves CE to /MyBoxD
PREV	Moves CE to /MyRtorA
PREV 2	Moves CE to /MyCylB
PREV BOX	Moves CE to /MyBoxA
PREV 2 CYL	Moves CE to /MyCylA
FIRST	Moves CE to /MyBoxA
FIRST 2	Moves CE to /MyCylA
FIRST CYL	Moves CE to /MyCylA
FIRST 3 CYL	Moves CE to /MyCylC
BOX 2	This is the same as FIRST 2 BOX. Moves CE to /MyBoxB.
LAST	Moves CE to /MyBoxD
LAST 2	Moves CE to /MyCylD
LAST CYL	Moves CE to /MyCylD
LAST 3 CYL	Moves CE to /MyCylB

2.9.3 Move to Next Lower Level

The syntax for moving down a level shares much of the syntax for moving within the level.

Int	descend to nth child
First Member	Goes to 1st member
Last Member	Goes to last member
First <element type>, FirstMember<element type>	Goes to first member of given type
First int <element type>	Go to nth element of given type in members list
Last <element type>, LastMember<element type>	Goes to last member of given type
Last int <element type>	Go to nth last element of given type in members list
Next <element type>	Goes to next element in member list from current position
Next int <element type>	Goes to next nth element in member list of given type from current position
Prev <element type>	Goes to next element in member list from current position
Prev int <element type>	Goes to previous nth element in member list of given type from current position
<element type> int	descend to nth child of given type

Example

We can use the same example as before but in this case we are positioned at the owning equipment, say /MYEQUI. The current position is defaulted to the start of the list. The member list being:

1 BOX /MyBoxA
2 CYL /MyCylA
3 CYL /MyCylB
4 RTOR /MyRtorA
5 CYL /MyCylC
6 BOX /MyBoxB
7 BOX /MyBoxC
8 CYL /MyCylD
9 BOX /MyBoxD

5	Moves CE to /MyCylC (5th member)
FIRST MEMBER	Moves CE to /MyBoxA
LAST MEMBER	Moves CE to /MyBoxD
FIRST CYL	Moves CE to /MyCylA
FIRST 3 CYL	Moves CE to /MyCylC

LAST CYL	Moves CE to /MyCylD
LAST 2 CYL	Moves CE to /MyCylC
NEXT CYL	Moves CE to /MyCylA (same as FIRST CYL)
NEXT 2 CYL	Moves CE to /MyCylB (same as FIRST 2 CYL)
PREV CYL	Invalid as there are no cylinders before the current position
BOX 4	Moves CE to /MYBoxD

In the above examples, the use of NEXT had the same result as using FIRST. The use of PREV was invalid. This is because the current position was off the start. We can change the current position using the END syntax to give more meaningful examples

e.g.

```
/MyCylB
END
```

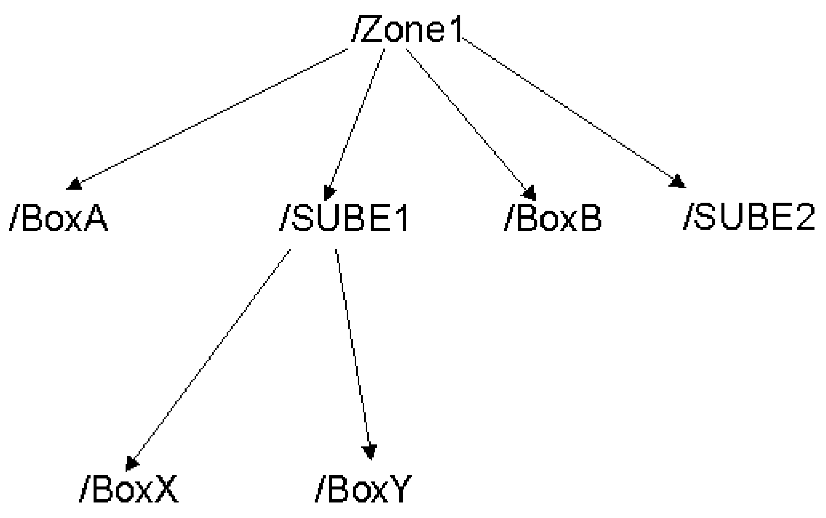
The CE is /MyEqui as before, but with the current position at /MyCylB

NEXT CYL	Moves CE to /MyCylC
NEXT 2 CYL	Moves CE to /MyCylD
PREV CYL	Moves CE to /MyCylA

2.10 Syntax Ambiguity Between Moving Across and Down

In most cases there is no ambiguity with having some of the same syntax for moving down and moving across. This is because it is rare to have the same element type as a sibling and a member. However there are some situations where this does occur. In these cases, the default is to move down rather than across.

e.g.



If the CE is /SUBE1, then

BOX 1	Moves CE to /BoxX (NOT /BoxA)
NEXT BOX	Moves CE to /BoxX (NOT /BoxB)
LAST BOX	Moves CE to /BoxY (NOT /BoxB)

2.11 Climbing up by Default

The commands to move to an element at the same level, may also be used for elements at any higher level. i.e. if the command is invalid at the CE, Outfitting will keep on climbing until the command becomes valid.

e.g. for the previous example, with the CE at /BoxY:

SUBE 2	Moves CE to /SUBE2
LAST SUBE	Moves CE to /SUBE2
FIRST ZONE	Moves CE to /Zone1 (assuming that this is the first zone)

2.12 Using the ‘OF’ Syntax

The commands described so far all work on the CE. It is allow able to do a navigation relative to any element by using the ‘OF’ syntax.

e.g. FIRST MEMBER OF /ZONE1

The ‘constructed’ name is actually an example of the use of the ‘OF’ syntax.

The ‘OF’ syntax can be nested as much as required.

e.g. FIRST MEMBER OF FIRST BOX OF NEXT EQUI

2.13 Other Syntax

2.13.1 Using the GOTO Syntax

The GOTO command can be used to go to any reference attribute. E.g.

GOTO CREF This will go to the element pointed to by the CREF of the CE.

As with other navigation commands, the ‘OF’ syntax may be used to go to reference on a different element.

e.g GOTO HREF OF /BRANCH88

Pseudo attributes can be specified after GOTO. A particularly useful pseudo attribute is FRSTW. This goes to first world of a given type.

e.g. GOTO FRSTW CATA

This will go to the first catalogue world.

2.13.2 Returning to the Previous Current Element

The SAME command will always return to the previous current element.

e.g.

```
/VESS1
/SECTION99
SAME
```

The current element will now be /VESS1.

2.14 ID Expressions

The above commands are all examples of an ID(identification) expression. The one exception is the ‘GOTO’ syntax. This keyword is omitted within an ID expression. ID expressions should be enclosed in brackets, although in most cases, they will work without the brackets. An ID expression may itself be queried or assigned to a PML variable.

Querying an expression or assigning it to a PML variable dos NOT change the CE.

```
Q ( NEXT BOX)
!MyEle = ( next box )
!MyEle = ( next box of /VESS1 )
!MyEle = (SPRE )
```

Assigning an ID expression to a PML variable is a common way to write PML.

2.15 Special Cases

2.15.1 UDETs

A UDET may be used wherever an element type is valid.

e.g.

```
(:MYBOX 1 OF /VESS1 )
NEXT :MYBOX
FIRST 2 :MYBOX
```

The following exception applies:

- When climbing, either the UDET or the base type may be specified.

  e.g.

  At a BOX below /VESSEL which is a :MYEQUIP

  :MYEQUIP - will climb to /VESSEL

  EQUIP - will also climb to /VESSEL

2.15.2 Trace Command

If in TTY mode, the TRACE ON/OFF command can be turned on track changes in current element. The default is on.

2.16 Pseudo Attributes Relating to Navigation

The following pseudo attributes relate to the database hierarchy. These can be queried directly or via a PML variable.

Attribute Name	**Data Type**	**Qualifier**	**Description**
ALLELE	DBREF	ElementType	All elements in the MDB of a particular type
CONNECTIONS	DBREF array		Connections
CONNECTIONSH	DBREF array		Connections for all descendants
CONNER	String	Int	Connection error message
DDEP	Int		Database depth within hierarchy (World is 0)
FRSTW	DBREF	String	Reference of first world of given DB type in current MDB
MAXD	Int		DB hierarchy depth of lowest level item beneath element
MBACK	DBREF array	*ElementType	Members in reverse order
MCOU	Int	*ElementType	Number of Element Members of Given type
MEMB	DBREF array	*ElementType	All members, or members of specific type
OWNLST	DBREF array		Owning hierarchy
PARENT	DBREF	*ElementType	Reference of ascendant element of specified type
SEQU	Int		Sequence Position in Member List
TYSEQU	Int		Type Sequence Number

'*'- qualifier is optional

3 Attributes

3.1 PML Attribute Class

3.1.1 Creation

A PML attribute instance may be created for a system attribute or a UDA.

e.g.

```
!AXLEN = object attribute('XLEN')
!UINT = object attribute(':UINT')
```

The class should not be confused with the attribute *value*. The actual Attribute *value* for a particular Element can only be accessed via the DBREF class or via the QUERY command. Comparing two Attributes just compares whether they identify the same attribute, the comparison does not look at attribute values in any way.

3.1.2 Methods

The Attribute instance can then be used for querying the 'meta data'. i.e. data about data. The methods of the class allow the following to be queried.

String Type()	Attribute type
String Name()	Attribute name
String Description()	Attribute description
Int Hash()	Attribute hash value
int Length()	Attribute length
bool IsPseudo()	Whether pseudo or not
bool IsUda()	Whether a UDA or not
string querytext()	As output at the command line when querying attribute
string units	Either BORE, DISTANCE or NONE
bool Noclaim()	Whether attribute can be changed without doing a claim
ElementType array ElementTypes	List of Elements for which the attribute is valid. This only works for UDAs
Real array limits	Min/Max values for real/int types
String array ValidValues(ElementType)	List of valid for text attributes. The list may vary with element type

string DefaultValue (ElementType)	Attribute default. This only works for UDAs
string Category()	Determines the grouping of attributes on the ‘Attribute Utility’ form
bool hyperlink()	if true then the attribute value refers to an external file
bool connection()	if true then the attribute value will appear on the reference list form
bool hidden()	If true then attribute will not appear on the Attribute utility form or after ‘Q ATT’
bool protected()	if true then attribute is not visible if a protected DB

Note: We do yet not support direct usage of this class in other syntax.

3.1.3 Attribute Type

Attributes can be the following type:

INT
REAL
LOGICAL
TEXT
REFERENCE
POSITION
DIRECTION
ORIENTATION
WORD

3.2 PML ElementType Class

3.2.1 Creation

An ElementType instance may be created for a system Element type or a UDET.

e.g.

```
!EQUI = object elementtype('EQUI')
!UEQUI = object elementtype(':MYEQUI')
```

The ElementType instance can then be used for querying the ‘meta data’. i.e. data about data. The methods of the class allow the following to be queried.

3.2.2 Methods

string Name()	Name of element type
string Description()	Description of element type
int Hash()	Hash value
bool IsUdet()	Whether a UDET or not

Attribute array systemAttributes()	List of system attributes (excludes UDAs)
string array DbType()s	List of valid DB types
string ChangeType()	Indicates if an element of this type may have it's type changed
ElementType SystemType()	for UDETs this is the base type
ElementType array	UDETs derived from this type
bool Primary()	Whether the element is primary or not
ElementType array MemberTypes()	Valid members, including UDETs
ElementType array ParentTypes()	Valid parents, including UDETs

Note: We do yet not support direct usage of this class in other syntax.

3.2.3 Related Pseudo Attributes

There are a number of pseudo attributes that return values according to the element type, as follows:

Attribute Name	Data Type	Qualifier	Description
HLIS	WORD(2000)		List of all possible types in owning hierarchy
LIST	WORD(2000)		List of all possible member types
LLIS	WORD(2000)		List of all possible types in member hierarchy
OLIS	WORD(2000)		List of all possible owner types
REPTXT	STRING		Reporter text used for element type

3.3 Querying Attributes

3.3.1 Querying the List of Attributes

The attributes available for an element will depend on its type. E.g. a site will have different attributes to a branch. The lists of valid attributes can be obtained as follows:

1. The ATTLIS attribute returns the list of all non pseudo attributes for that element. This list includes UDAs and hidden attributes. The same list is used for the PML DBREF ATTRIBUTES method.
2. The ATTOUT attributes is as for ATTLIS but excludes hidden attributes. This list is used when doing a 'Q ATT' and by the attribute utility form. Attributes that are hidden can still be queried individually in the normal way.
3. The valid UDAs can be queried using the UDALIS attribute. This includes hidden UDAs.
4. The PSATTS attribute returns the list of valid pseudo attributes. Typically this list is large, running into the hundreds. N.B. querying PSATTS can be slow.

3.3.2 Standard Attribute Query

An attribute value may be obtained as follows:

Specify the attribute name after QUERY or on the RHS of a PML assignment. This will return the attribute value, if valid, for the CE.

e.g.

```
Q XLEN
!A = XLEN
```

Via a DBREF object using the attribute name as a method, or using the ‘Attribute’ method.

e.g.

```
Q var !!CE.XLEN
!A = !!CE.ATTRIBUTE(‘XLEN’)
```

The querying is the same for UDAs or pseudo attributes.

e.g.

```
!RESULT = !!CE. ATTRIBUTE(‘ATTLIS’)
!RESULT = !!CE. ATTRIBUTE(‘:MYUDA’)
```

The type of the PML variable will depend on the type of the attribute and whether it is an array or not. Attributes of INTEGER type will be assigned to a PML variable of type REAL. Attributes of type WORD are assigned to PML variable of type TEXT.

If the attribute is a DISTANCE attribute and current DISTANCE units are inch or finch, then the value will be converted to inches. If the attribute is a BORE attribute and current BORE units are inch or finch, then the value will be converted to inches.

3.4 PML1 Syntax

PML1 syntax allows an attribute to be passed to a PML variable without the ‘=’ operator. If this is done then the value will always be formatted to a TEXT using the current units if applicable.

e.g.

```
VAR !RESULT XLEN
!RESULT will be of type TEXT
```

3.5 Querying Arrays

If the attribute is an array, the query will return a list of values. Individual elements can be queried by passing in the index number.

e.g.

```
!VALUE = !!CE.DESP[2]
!VALUE = DESP[2]
```

Alternatively, the NUM keyword can be used for PML 1 syntax. A range of values can be returned using the TO keyword.

e.g.

```
VAR !VALUE DESP NUM 3 - to retrieve the 3rd value
VAR !VALUE DESP NUM 3 TO 5 - to retrieve 3rd to 5th values
Q DESP NUM 3 TO 5
```

An error will occur if attempting to query off the end of the array.

Within a PML1 expression, a position attribute may be queried as an array in order to access the individual coordinates.

e.g.

```
VAR !X  (POS[1] ) - This will return the X coordinate.
```

3.5.1 Using the OF Syntax

Attributes my be queried on other elements by using the ‘OF’ syntax.

e.g.

```
Q XLEN of /MYBOX
!RESULT = XLEN OF /MYBOX
```

The syntax following the OF may be any ID expression.

e.g.

```
!RESULT = XLEN of NEXT BOX
!RESULT = DESC OF CREF.
```

3.5.2 dot Notation in PML

For reference attributes, the PML dot notation can be used to achieve a similar result.

e.g.

```
!RESULT = !!CE.ATTRIBUTE(‘CREF’).DESC
```

This will return the description of the element pointed to by the CREF attribute on the CE.

3.5.3 Qualifier

Many pseudo attributes take a qualifier. The qualifier is the extra information required to make the query. Examples of where a qualifier is used are:

1. Querying a ppoint position (PPOS) requires the ppoint number
2. The ATTMOD attribute can be used to query when an attribute was modified. The qualifier is the attribute to test for modification.
3. A direction/position may be queried wrt another element.

The definition of what pseudo attributes take what qualifier is described in the data model reference manual.

The qualifier follows the attribute name in brackets. Attribute qualifiers must be preceded by the keyword ATTNAME and element types must be preceded by the keyword TYPENAME.

e.g.

```
Q PPOS(1) - PPOS has an int qualifier
Q LASTMOD(ATTNAME XLEN) - LASTMOD has an attribute qualifier
Q MEMBER(TYPENAME BOX) - MEMBER has an optional element type qualifier
```

For PML variables, the qualifier should be assigned to a PML array object and passed to the ‘Attribute’ method as the second argument:

e.g. to query PPOS 1

```
!q=object array()
!q[1] = 1
q var !!ce.attribute('PPOS', !q)
```

e.g. to query list of nominal bores:

```
!q=object array()
!q[1] = 'BORE'
q var !!ce.attribute('NOMBMM', !q)
```

e.g. to query Equipment members:

```
!q=object array()
!q[1] = object elementtype('EQUI')
q var !!ce.attribute('MEMBER’, !q)
```

3.5.4 Relative Positions, Directions, Orientations

Positions, Orientations, directions can be queried relative to another element using the WRT syntax.

e.g.

```
Q POS WRT /MYBRAN
Q PPOS(1) WRT /MYBRAN
```

The use of WRT is described more fully in the *Expressions*.

3.5.5 Summary of Related Pseudo Attributes

Pseudo attributes relating to the list of attributes.

Attribute Name	Data Type	Qualifier	Description
ATTLIST	WORD(300)		List of all visible attributes for element
PSATTS	WORD(500)		List of pseudo attributes
UDALIS	WORD(300)		List of UDAs
UDASET	WORD(300)		List of UDAs set

Pseudo attributes relating to the name

Attribute Name	Data Type	Qualifier	Description
CUTNAM	STRING(700)	NUMBER	Full name of element, truncated to n characters
CUTNMN	STRING(700)	NUMBER	Full name of element (without leading slash) truncated to n characters

Attribute Name	Data Type	Qualifier	Description
FLNM	STRING(700)		Full name of the element
FLNN	STRING(700)		Full name of the element (without leading slash)
ISNAMED	BOOL		True if element is named
NAMESQ	STRING(700)		Type. sequence number and name of element
NAMETY	STRING(700)		Type and name of the element
NAMN	STRING(500)		Name of the element (without leading slash)
NAMTYP	STRING(700)		Type and full name of element

Pseudo attributes relating to the type

Attribute Name	Data Type	Qualifier	Description
ACTTYPE	WORD		Type of element, truncating non UDETs to 4 or 6 characters
AHLIS	WORD(200)		List of actual types in owning hierarchy
OSTYPE	WORD		Short cut for "Type of owner"
TYPE	WORD		Type of the element, ignoring UDET, truncated to 4 or 6 characters

3.6 Setting Attributes

3.6.1 Standard Syntax

Attribute values can be set in two ways:

1. By assigning a value via a PML variable e.g. !!CE.XLEN = 99

Note: There must be a space between the ‘=’ and a digit. “!!CE.XLEN =99” would not be valid.

2. Use the attribute name to assign a value to the CE e.g. XLEN 99

The following general rules must be followed:

- The value assigned must be the correct type for the attribute type (see examples below)
- PML variables can not be directly used if using method (2). The PML variable must be expanded using the late evaluation syntax, i.e. ‘XLEN !A’ is invalid but ‘XLEN $!A’ is OK. This also applies to any PML variables within expressions.

The behaviour for each attribute type is described below:

REAL attribute - allows an int, real or real expression

e.g.

```
!A = 1000
!!CE.XLEN= !A
!!CE.XLEN= (99.9 + !A )
XLEN $!A
XLEN (99.9 + $!A )
XLEN (99 + XLEN OF PREV BOX )
```

INTEGER attribute - allows an int, a real or real expression. The result will be rounded to the nearest integer.

e.g.

```
!!CE.AREA = 99.6
Q AREA – will now return 100
```

TEXT attribute - allows a text value, a text expression, or UNSET. Assigning UNSET will result in a zero length string.

e.g.

```
!A = ‘Some text ‘
!!CE.DESC = ‘My description’
!!CE.DESC = (!A + ‘extra text’)
DESC UNSET
```

LOGICAL attribute - allows FALSE, TRUE or logical expression.

e.g.

```
SHOP TRUE
!A = 99
!B = 100
!!CE.SHOP ( !A GT !B)
SHOP ( $!A GT $!B)
```

REF attribute - allows a name, refno , ID expression, or UNSET, NULREF keywords. The UNSET and NULREF keywords both result in a null reference (=0/0) being assigned.

```
CREF =123/456
CREF /MYBRAN
CREF UNSET
CREF NULREF
!!CE.CREF (FIRST MEMBER OF /PIPE1 )
```

Note: There must be a space between the name and the ‘)’

WORD attribute - If assigning to a PML variable, then allows a text value or text expression. e.g.

```
!A = ‘FLG’
!!CE.TCON = !A + ‘D’
```

If assigning via the attribute name, then it must be a word.

e.g. TCON FLGD

POSITION attribute - allows a position or position expression.

```
HPOS N 100 U 100
!!CE.POS = (N 100 from /MYEQUIP )
AT N 100 from /MYEQUIP
```

Note: The POS attribute can not be set by name, use AT syntax instead.
Do not use brackets if setting by attribute name.

DIRECTION attribute - Allows a direction or direction expression

```
HDIR N 45 U
HDIR TOWARDS /MYEQUIP
!!CE.HDIR = (TOWARDS /MYEQUIP )
```

Note: Do not use brackets if setting by attribute name.

ORIENTATION attribute - Allows an orientation or an orientation expression

```
ORI N IS U
!!CE.ORI = (N IS E WRT /VESS1 )
```

Note: Do not use brackets if setting by attribute name.

3.6.2 Setting a UDA Back to a Default

A UDA may be set back to it's default by using the DEFAULT keyword.

e.g. :MYUDA DEFAULT

3.6.3 Setting an Array

If assigning via a PML variable, an array attribute must be assigned from a PML array object.

e.g. !!CE.DESP = !MYARRAY

If assigning via the attribute name, then a list of values must be given.

e.g. DESP 1 2 3 4 5

3.6.4 Single Value of an Array

If assigning via a PML variable, an index number may be specified in square brackets.

e.g. !!CE.DESP[2] = 99

If assigning via the attribute name, a single value of an array may be set using the NUMB keyword. The NUMB keyword follows the attribute name, and is followed by the index number.

e.g. DESP NUMB 2 99

This sets the 2nd value of the array to 99.

The NUMB command actually specifies the start point for a list of values.

e.g. DESP NUM 3 99 100 101

This would set the 3rd value to 99, the 4th to 100 and the 5th to 101.

The new values may go off the end of the existing array, but the start point must not be more than one beyond the existing end point.

e.g.

DESP 1 2 3 - set up initial values
DESP NUMB 4 99 - OK, as at end
DESP NUMB 6 100 - Error, as would leave a gap

3.6.5 Special Syntax for Names

Naming Design Elements

All elements except the WORLD can be named. Although Design elements are often given suitable names while being created, later name changes can be made by giving a new name or by removing the old name. The name of any element must be unique; that is, not already used for another currently accessible element.

Examples:

`NAME /ZONE5D`	The current element is given the specified name provided it has not been used elsewhere.
`UNN`	The current element loses its name (it is still identifiable by its automatically allocated reference number).

Command Syntax:

```
>-- NAMe --+-- ALL name name --.
           |                   |
           '-- name -----------+-->
>-- UNName -->
```

Renaming Elements and Their Offspring

The name of the current element and offspring can be modified where a standard name part occurs.

Example:

`REN ALL /Z1 /Z2`	All occurrences of /Z1 in the names of the current element and its offspring will be changed to /Z2.

Command Syntax:

```
>-- REName --+-- ALL name name --.
             |                   |
             '-- name -----------+-->
```

3.6.6 Special Syntax for LOCK

Locking Elements Against Alteration and Deletion

Locking a design element prevents it from being modified or deleted. The LOCK command allows either a single element to be controlled, or all its offspring too. (A complete Site can be locked if required.) This provides you with personal security control over your area of work. (General security restrictions affecting the whole Project are established in the ADMINISTRATION module of Outfitting.)

Examples:

`LOCK ALL`	The current element and all its offspring are locked.
`UNLOCK`	The current element is unlocked.

Command Syntax:

```
>--+-- LOCK ----.
   |            |
   '-- UNLOck --+-- ALL --.
                |         |
                '---------+-- <snoun> --.
                          |             |
                          '-------------+-->
```

3.6.7 Related Pseudo Attributes

Attribute Name	Data Type	Qualifier	Description
DACMOD	BOOL	ATTR	True if DAC allows attribute of element to be modified
MODATT	BOOL	ATTR	True if attribute of element can be modified
MODERR	STRING(120)	ATTR	Returns the error text that would occur if attribute was modified

4 Database Modification

This chapter describes the commands to create, copy and modify database elements.

4.1 Modifying the Content of a DB

As well as accessing the current content of a DB, you may also (if you have Read/Write access rights) modify a DB in any of the following ways:

- Create a new element at an appropriate level of the DB hierarchy; see *Creating a New Element*
- Delete an element from the DB hierarchy; see *Deleting an Element*
- Reorganise the hierarchy by rearranging members of an element into a different list order or
  by moving an element from one part of the hierarchy to another; see *Reorganising the DB Hierarchy*
- Define the attributes and offspring of a new element by copying the corresponding attribute
  settings and member lists from another element; see *Copying Attributes from One Element to Another*

4.1.1 Creating a New Element

To create a new element within an existing DB, you must first ensure that the Current Element is at a level within the hierarchy which can legally own the element to be created. For example, a Site can own a Zone, but it cannot own a Valve. To create a new Valve, you *must* be at Branch level. You must therefore navigate to the correct level by using one of the command options described in *Database Navigation and Query Syntax*.

Note: The Q LIST query will tell you which element types you can create as members of the Current Element.

You can then create a new element, set its attributes and, if required, create further elements as its members.

Creating an Element After the Current List Position

If you create an element without explicitly identifying its position in the Member List of the Current Element, the new element is inserted immediately *after* the Current List Position. To use this option, enter the command

NEW *element_type element_name* (*element_name* is optional)

For example, if the Current List Position is at member 4 (/VALV1) of the Member List.

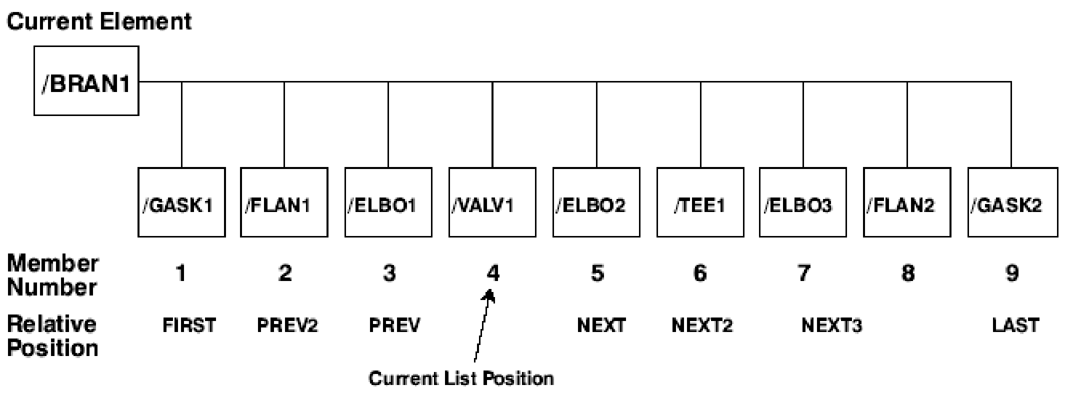


Figure 4:1. Current Element and its Member List (illustrating movement along list)

the command

```
NEW TEE /TEE2
```

adds a new Tee at list position 5 (between /VALV1 and /ELBO2) and names it /TEE2. The Member List of /BRAN1 thus becomes

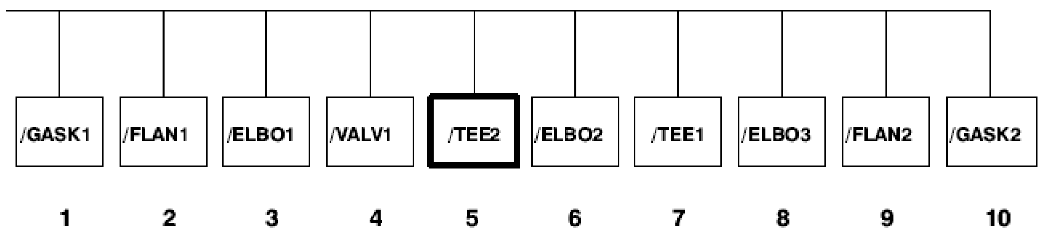


Figure 4:2. Result of adding a new Tee

To insert the new Tee as the first or last component in the Member List, access the Branch Head or Tail, respectively, before giving the NEW TEE command.

Creating an Element at a Specified List Position

To create a new element at a specified list position, identify a list position adjacent to the required position and state which side of it the newly-created element is to go. The command syntax is one of the following:

NEW *element_type* *element_name* **BEFore** *list_position*

NEW *element_type* *element_name* **AFTer** *list_position*

where *element_name* is again optional and where *list_position* may be specified in any of the ways described in Database Navigation and Query Syntax.

Consider the following examples. Starting from the configuration shown, any of these commands creates a new Tee between /ELBO3 (list position 7) and /FLAN2 (list position 8):

`NEW TEE AFTER /ELBO3`	Specify name or refno
`NEW TEE BEFORE 8`	Specify list position number
`NEW TEE BEFORE FLAN 2`	Specify member type and number (second Flange in the list)

`NEW TEE AFTER LAST ELBO`	Specify first or last member of a given type (last Elbow in the list)
`NEW TEE AFTER NEXT 3`	Specify position relative to Current List Position
`NEW TEE BEFORE LAST FLAN`	Specify first or last member of a given type

The new Tee, which is unnamed, becomes list member 7, /ELBO3 becomes list member 8, /FLAN2 becomes list member 9, and so on.

4.1.2 Deleting an Element

You can delete either the entire Current Element or some or all of its offspring. When you delete the Current Element, you also delete *all* of its offspring (that is, its members, their members, etc.) from the hierarchy. The command must therefore be used with care. When an element has been deleted, its Owner becomes the new Current Element.

As a safeguard against accidental deletion of parts of a DB, the deletion function operates only on the Current Element. As further safeguards, the **DELETE** command word must be entered in full and the command syntax requires that you confirm the generic type of the Current Element. Furthermore, access to the required element and its subsequent deletion must be specified in two separate command lines.

To **delete** the **Current Element** and all its **offspring**, enter

DELETE *element _type*

For example, to delete a Nozzle, make the Nozzle the Current Element and then enter

```
DELETE NOZZ
```

The Equipment which owned the Nozzle becomes the Current Element.

To delete a complete Zone, including all Equipment, Piping, Structures etc. owned by it, make the Zone the Current Element and then enter

```
DELETE ZONE
```

The Site which owned the deleted Zone becomes the Current Element.

To **delete** only specified **members** of the Current Element, use one of the following forms of the command syntax:

DELETE *element_type* **MEMbers**	(deletes all members)
DELETE *element_type* **MEMbers** *integer*	(deletes one member)
DELETE *element_type* **MEMbers** *integer* **TO** *integer*	(deletes a range of members)

Consider the following examples, where the Current Element is /BRAN1 with the Member List illustrated in Figure 10-2:

`DELETE BRAN MEMBERS`	Deletes all components from the Branch, leaving only the Branch Head and Tail
`DELETE BRAN MEMBER 6`	Deletes only /TEE1
`DELETE BRAN MEMBERS 5 TO 7`	Deletes /ELBO2, /TEE1 and /ELBO3

4.1.3 Reorganising the DB Hierarchy

You can reorganise the structure of the DB hierarchy, without elements being added to or removed from its contents, in either of two ways:

- By rearranging the order of the Member List of a single element
- By relocating an element to a different part of the hierarchy

In both cases elements and their offspring are transferred to new positions in the hierarchy. In the first case the element's owner remains unchanged, while in the second case the element's owner changes.

To **rearrange** the **Member List** of the Current Element, use one of the commands:

REOrder *element_id*
REOrder *element_id* **BEFore** *list_position*
REOrder *element_id* **AFTer** *list_position*

where *element_id* specifies an element which is to be moved (which must be a member of the Current Element) and where *list_position* may be specified in any of the ways described in *Database Navigation and Query Syntax*.

If *list_position* is omitted, the intended position is assumed to be immediately *after* the Current List Position.

For example, starting with the previous Member List:

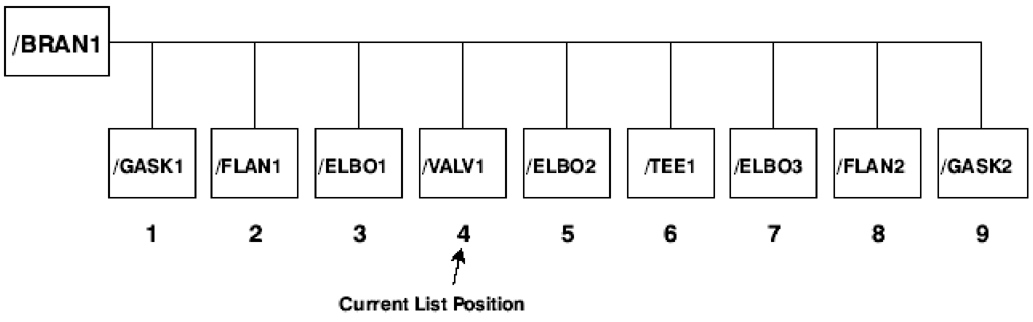


Figure 4:3. Example Member List

the command

```
REORDER /ELBO3
```

moves /ELBO3 to position 5, immediately following the Current List Position, giving the new Member List

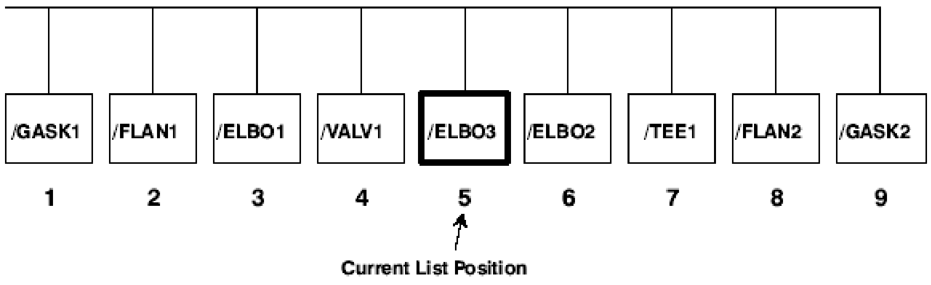


Figure 4:4. Example of REORDER

Starting from *either* of the above configurations, the command

```
REORDER /ELBO3 BEFORE FIRST ELBO
```

moves /ELBO3 to position 3, immediately before /ELBO1, thus

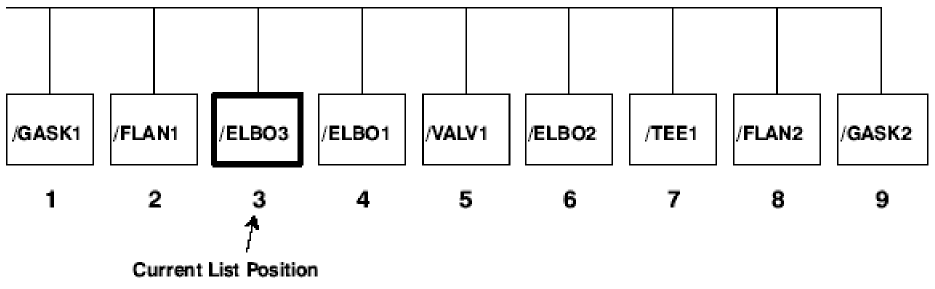


Figure 4:5. Example of REORDER

To **insert** an existing element **into** the **Member List** of the Current Element, when it is not already a member of that list, use one of the commands

INCLude *element_id*

INCLude *element_id* **BEFore** *list_position*

INCLude *element_id* **AFTer** *list_position*

where *element_id* specifies an element which is to be moved (which may be anywhere within the DB hierarchy as long as it is at an appropriate level) and where *list_position* may be specified in any of the ways described in *Database Navigation and Query Syntax*.

If *list_position* is omitted, the intended position is assumed to be immediately *after* the Current List Position.

For example, starting with the simple hierarchy

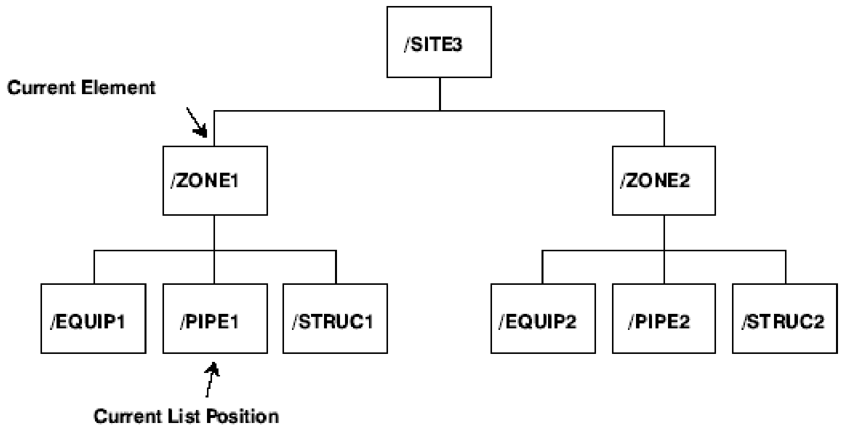


Figure 4:6. Example Hierarchy

the command

```
INCLUDE /PIPE2
```

moves /PIPE2 (and all its offspring) to the position immediately following the Current List Position. Ownership of /PIPE2 passes from /ZONE2 to /ZONE1, resulting in the new hierarchy

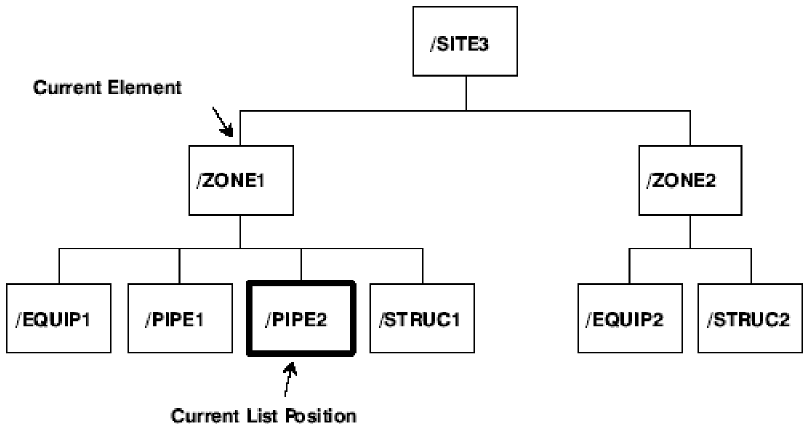


Figure 4:7. Example Hierarchy after INCLUDE /PIPE2 command

4.1.4 Copying Attributes from One Element to Another

It is often convenient to create a new element as a copy of an existing element which has the same, or similar, attribute settings or members to those required. You do this in two stages:

1. Create a new element (as described in *Creating a New Element*), which becomes the Current Element.
2. Copy the attributes of another element (the 'source' element) so that they also become the attributes of the newly created Current Element (the 'target' element). The existing attribute settings, usually the defaults, are overwritten by the copied settings.

When an element is 'cloned' in this way, all attributes are copied from the source element to the target element except NAME (which must be unique) and LOCK (which is always *unlocked* in the target element). Additionally, and this is what makes the facility so powerful, all offspring of the source element are copied as offspring of the target element.

Note: If the Current Element already has members, it is *not* possible to make it a copy of another element in this way.

You may specify automatic **renaming** of the Current Element and its offspring as part of the copying process. Without this the new elements will be unnamed, since Outfitting does not permit two elements in the same DB hierarchy to have identical names. You may also choose to copy only the **members** (and their offspring) of the source element, leaving the attributes of the Current Element itself unchanged.

To **copy a complete element** and all of its offspring, after creating a new Current Element of an appropriate type, enter

COPY *element_id*

where *element_id* identifies the source element to be copied.

For example, to create a new item of Equipment which is an exact replica of a previously-defined Equipment, you might use the command sequence (at Zone level)

```
NEW EQUI /EQUIPB
COPY /EQUIPA
```

This creates /EQUIPB as the Current Element and then turns it into an exact copy of /EQUIPA. All attributes and members of /EQUIPB now have the same settings as those of /EQUIPA, including its position, orientation etc., and so you will probably now want to move one of the Equipments to a different location.

To **copy all offspring** of an element, so that they create duplicate offspring for the Current Element, enter

COPY MEMbers OF *element_id*

The position, orientation, etc., of the Current Element now remain unchanged, but it acquires new members which are derived from the specified source element and which are correctly positioned relative to the Current Element.

To **copy selected offspring** of an element, so that they create duplicate offspring for the Current Element, enter

COPY MEMbers *integer* **TO** *integer* **OF** *element_id*

For example, the command sequence

```
NEW BRAN /SIDEARM
COPY MEMBERS 12 TO 20 OF /MAINLINE
```

creates a new Branch named /SIDEARM whose components replicate that part of the existing Branch /MAINLINE between the specified list positions. The attributes of the Branch /SIDEARM itself are unaffected by the **COPY** command, so that its position, orientation, etc. (as defined by its Head and Tail settings) remain unchanged by the addition of its new members.

To **copy attributes from an identified element into the current element**, type

COPY ATTributes OF *element_id*

This causes all attributes (except for references to elements in DESI databases and OWNER) to be copied to the current element. Or:

COPY LIKE *element_id*

This is similar to the ATTRIBUTES option, except that as well as DESI references not being copied, neither are any position, direction, orientation or angle attributes.

In both cases, the SPREF and CATREF are also not copied between elements of different types.

To **copy elements alongside their original positions**, type

COPY ADJ/ACENT *select*

This option causes a list of elements, defined by the selection criterion *select,* to be copied alongside their original positions in the database. So if the list includes a SCTN and a PNOD (for example) then each of these items would be copied so that the new SCTN shares the same owner as the old SCTN and the new PNOD shares the same owner as the old PNOD. As this option copies elements, rather than just attributes, other COPY options, such as RENAME, are valid.

To copy all or part of an element and rename the copies, append the command

... REName *old_name* *new_name*

to the corresponding COPY command line.

For example, the command

```
    COPY   /FRACT1/PIPE   RENAME   /FRACT1   /FRACT2
```

copies all attributes and offspring of /FRACT1/PIPE into the Current Element. Where /FRACT1 occurs as the name or part of the name, it is changed to /FRACT2 in the Current Element and its offspring. Thus the Current Element itself is now named /FRACT2/PIPE, and so on.

Related Pseudo Attributes

Attribute Name	**Data Type**	**Qualifier**	**Description**
DACCOH	BOOL		True if DAC allows element hierarchy to be copied to another DB
DACCOP	BOOL		True if DAC allows element to be copied to another DB
DACCRE	BOOL	NOUN	True if DAC allows element to be created
DACDEL	BOOL		True if DAC allows element to be deleted
DACERR	STRING(120)	ATTR	Returns the DAC error
MODATT	BOOL	ATTR	True if attribute of element can be modified
MODDEL	BOOL	ATTR	True if element can be deleted

5 Save Work and Get Work

SAVEWORK saves the current DESIGN changes without leaving DESIGN. It is good practice to use this command on a regular basis during a long session to ensure maximum data security.

As well as a comment, an optional number *n* can be used to specify a particular database for the command. The number is the number of the database in the order output by the STATUS command (see *Project*). If no number is given, the SAVEWORK applies to the whole MDB. An example of Savework syntax is SAVEWORK ‘comment’ 1.

GETWORK refreshes the view of all READ databases to pick up any changes that other users may have made since you first opened them. The optional *n* works in the same way as for SAVEWORK. You would normally only use GETWORK if you know of specific changes you wish to pick up and use. Please note that GETWORK slows up subsequent database access, as the information has to be re-read from disk. Therefore, you should use this command sparingly.

5.1 Sessions

Each time you enter DESIGN or save your design changes, a new session is created for each database changed. You can then query when specific items of design data were modified by reference to the corresponding session number(s). Sessions can be used by the System Administrator to backtrack changes to a given date or session if necessary.

5.2 Session Comments

You can add a comment for each session, which can help identify the work done in each session.

Lets you associate comment text with the current DESIGN session. You can query this text later to help you identify a particular session in which modifications were made to elements and/or attribute settings. You can enter the session comment *before* you issue a SAVEWORK command, or as part of a SAVEWORK command for example SAVEWORK ‘MY COMMENTS’.

Note: Sessions 1 and 2 are created in ADMIN (when the DESIGN DB and its World element, respectively, are created), so the first true session will be **Session 3**.

Example:

```
SESSION COMMENT ’Addition of upper platform’
```

Command Syntax:

```
>-- SESSION COMMENT -- text -->
```

Querying:

```
Q SESSComment integer
```

where *integer* is the session number.

Each time you enter DESIGN or save your design changes, a new session is created for each database changed. You can then query when specific items of design data were modified by reference to the corresponding session number(s). Sessions can be used by the System Administrator to backtrack changes to a given date or session if necessary.

6 Multiwrite Databases Claims and Extracts

If a DESIGN or Outfitting Draft DB has been created as a **multiwrite** database, several users can write to it simultaneously, although they cannot change the same element.

Multiwrite databases can either be **Standard** multiwrite databases, or **Extract** databases. In both types, an element must be **claimed** before it can be modified. Claiming an element prevents other users claiming (and modifying) the element; the element must be **released** before another user can change it.

Claiming can be either **explicit**, where the user must use the CLAIM command before attempting to modify the element, or **implicit**, where the claim is made automatically when the user tries to modify the element. The claim mode is set when the DB is created. For full details see the *Administrator Command Reference Manual*.

6.1 User Claims

In a Standard multiwrite database, you must claim an element before changing it. This is known as a **user claim**. If the claim mode is explicit (see below for details of how to check this), you must first claim each element that you want to modify using the CLAIM command. If the claim mode is implicit, the claim will be made automatically (although you can still give explicit CLAIM commands if you want to prevent other users claiming specific elements).

Only **primary** elements can be claimed, these are listed in the *Data Model Reference Manual*.

You can claim a specified element only, or a specified element plus all of the significant elements below it in the hierarchy. If the claimed element is not a significant element, the significant element above it in the hierarchy will be claimed.

An element must be **unclaimed** before another user can claim it and change it. User claims are always unclaimed when you change modules or leaves Outfitting, and you can also unclaim elements at any time during an Outfitting session using the UNCLAIM command.

Examples:

```
CLAIM /ZoneA /EQUIP100 /PIPE-100-A
```

Claims named elements

```
CLAIM /ZoneA HIERARCHY
```

Claims named element and all of its owned hierarchy

```
CLAIM /ELBOW-33
```

Examples:

Claims Branch which owns named component, since ELBO is not a significant element

```
UNCLAIM /PIPE-100 /PIPE-200
```

Unclaims named elements

```
UNCLAIM ALL
```

Unclaims all elements currently claimed

Command Syntax:

```
                .---------------.
               /                |
>-- CLAIM ----*-- elementname --+-- HIERARCHY ---.
                                |                |
                                '----------------+-->

                 .---------------.
                /                |
>-- UNCLAIM ---*-- elementname --+-- HIERARCHY ---.
               |                 |                |
               '-- ALL ----------+----------------+-->
```

6.1.1 Notes on Standard Multiwrite DBs

- Elements cannot be claimed if recent changes have been made to them by other users. You must issue a GETWORK command first.
- Elements cannot be unclaimed if there are updates outstanding. You must issue a SAVEWORK command first.
- You can insert/remove primary elements in a members list without claiming the owner. For example, you can add a Branch to a Pipe without claiming the Pipe. Thus two users can add different Branches to the same Pipe: any discrepancies will be resolved when a SAVEWORK is attempted.
- Before an element can be deleted, that element and all of its sub-hierarchy must be claimed.
- The following potential problems may not be revealed until you try to save changes:

1. If two concurrent users allocate the same name to different elements, the second user to attempt a SAVEWORK will show up an error. The second user must rename their element.
2. If one user inserts a significant element into another element’s list, while a concurrent user deletes the latter element, an attempt to SAVEWORK will show up an error. Either the first user must delete or move the significant element, or the second user must QUIT without saving the deletion.

6.1.2 Extract Databases

Unlike standard multiwrite databases, extracts allow users to keep elements claimed when they exit from Outfitting or change to another module. They can also be used, together with Data Access Control, to manage workflow. See the *Administrator User Guide* for more information.

An Extract is created from an existing Database. When an Extract is created, it will be empty, with pointers back to the owing or **master** database. Extracts can only be created from Multiwrite databases. An extract can be worked on by one User at the same time as another user is working on the master or another extract.

When a user works on the extract, an extract claim is made as well as a user claim.

If the claim mode is explicit, the extract claim will be made automatically when you make a user claim using the CLAIM command. You can also claim to the extract only using the EXTRACT CLAIM command.

If an element is claimed to an extract, only users with write access to the extract will be able to make a user claim and start work on the element:

- If the databases are set up with implicit claim, when the user modifies the element, the element will be claimed both to the extract and then to the user. If the element is already claimed to the extract, then the claim will only be made to the user.
- If the databases are set up with explicit claim, then the user will need to use the CLAIM command before modifying the element.
- Once a user has made a user claim, no other users will be able to work on the elements claimed, as in a normal multiwrite database.
- If a user unclaims an element, it will remain claimed to the extract until the extract claim is **released** or **issued**.

When an extract user does a SAVEWORK, the changed data will be saved to the Extract. The unchanged data will still be read via pointers back to the master DB. The changes made to the extract can be written back to the master, or dropped. Also, the extract can be refreshed with changes made to the master.

Examples:

```
EXTRACT CLAIM /STRU1 /STRU2 /STRU3
```

Claims named elements to the extract

```
EXTRACT CLAIM /STRU1 /STRU2 /ZONE-A HIERARCHY
```

Claims the named elements, and all the elements in the hierarchy to the extract

The `HIERARCHY` keyword must be the last on the command line. It will attempt to claim to the extract all members of the elements listed in the command which are not already claimed to the extract.

```
EXTRACT FLUSH DB PIPE/PIPE ‘Description of flush’
```

Writes all changes to the database back to the owing extract. The Extract claim is maintained.

```
EXTRACT FLUSH /STRU1 /STRU2 /STRU3 ‘Flushing three  structures’
```

Writes the changes to the named elements back to the owing extract. The Extract claim is maintained.

```
EXTRACT ISSUE DB PIPE/PIPE ‘Issuing /pipe’
```

Writes all the changes to the database back to the owning extract and releases the extract claim

```
EXTRACT ISSUE /ZONE-A HIERARCHY ‘Issuing /zone’
```

Examples:

Writes all the changes to the named element and all elements under it in the hierarchy back to the owning extract and releases the extract claim

```
EXTRACT ISSUE /STRU1 /STRU2 /STRU3 ‘Issuing three structures’
```

Writes the changes to the named elements back to the owning extract and releases the extract claim

```
EXTRACT RELEASE DB PIPE/PIPE
```

Releases the extract claim: this command can only be given to release changes that have already been flushed.

```
EXTRACT RELEASE /STRU1 /STRU2 /STRU3
```

Releases the extract claim: this command can only be given to release changes that have already been flushed.

```
EXTRACT RELEASE /ZONE-A HIERARCHY
```

Releases the extract claim to the named element and all: elements under it in the hierarchy.

```
EXTRACT DROP DB PIPE/PIPE ‘Dropping /Pipe’
```

Drops changes that have not been flushed or issued. The user claim must have been unclaimed before this command can be given.

```
EXTRACT REFRESH DB MYTEAMPIPING
```

This will refresh the extract MYTEAMPIPING with changes made on the parent extract,

The elements required can be specified by selection criteria, using a Programmable Macro Language (PML) expression. For example:

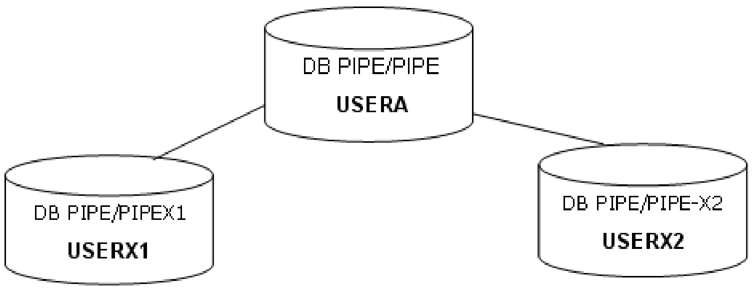
```
    EXTRACT CLAIM ALL STRU WHERE (:OWNER EQ 'USERA') HIERARCHY
```

Command Syntax:

```
>- EXTRACT -+- FLUSH ---------------.
            |                       |
            |- FLUSHWithoutrefresh -|
            |                       |
            |- RELEASE -------------|
            |                       |
            |- ISSUE ---------------|
            |                       |
            |- DROP ----------------|    .-------<-------.
            |                       |   /                |
            ‘- REFRESH -------------+--*-- elementname --+- HIERARCHY -.
                                    |                                  |
                                    |                                  |
                                    |                                  |
                                    ‘-- DB dbname ---------------------+->
```

6.1.3 How to Find Out What You Can Claim

This section explains what different users will see as a result of Q CLAIMLIST commands.



For this example, take the case of a database PIPE/PIPE, accessed by USERA, with two extracts. Users USERX1 and USERX2 are working on the extracts.

USERA creates a Pipe and flushes the database back to the owning database, PIPE/PIPE. The results of various Q CLAIMLIST commands by the three Users, together with the extract control commands which they have to give to make the new data available, are shown in the *Figure 6:1.: Querying extract claimlists*.

Note: Q CLAIMLIST EXTRACT
tells you what you can flush

Q CLAIMLIST OTHERS
tells you want you can't claim

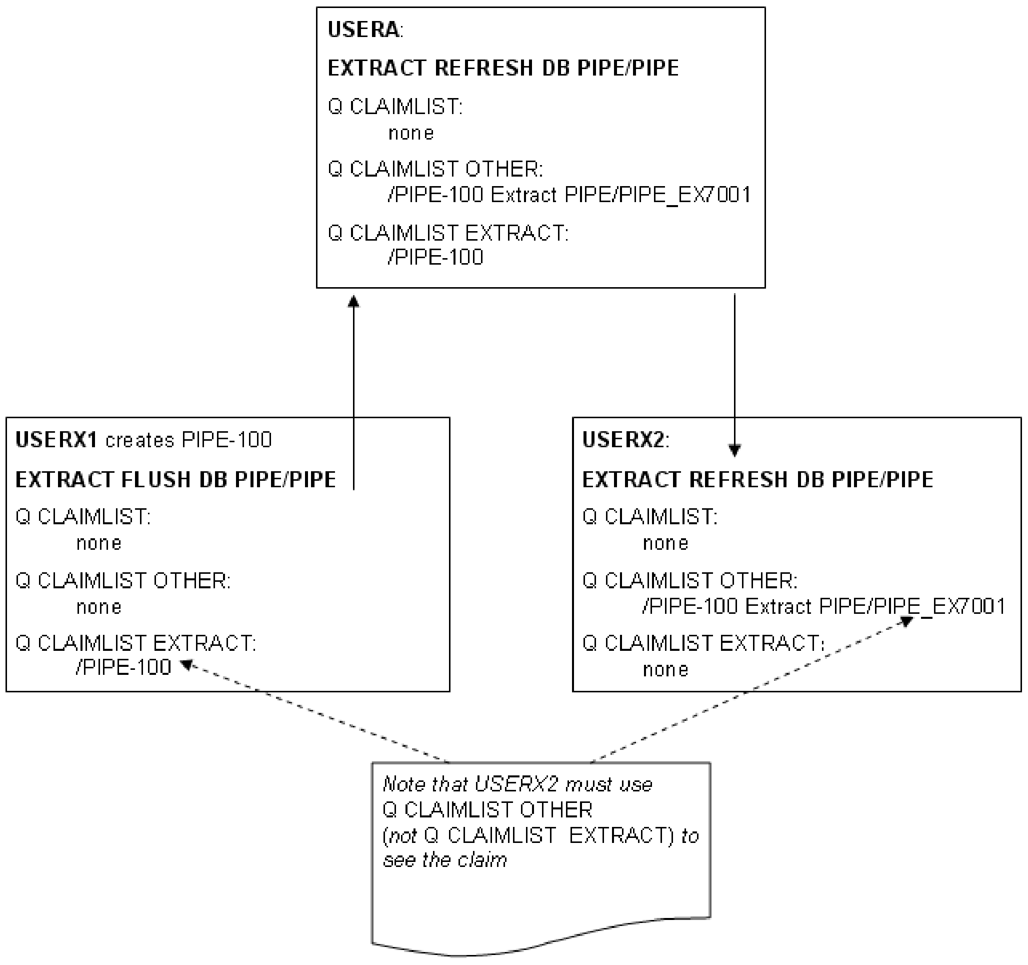

USERA:
EXTRACT REFRESH DB PIPE/PIPE
Q CLAIMLIST:
none
Q CLAIMLIST OTHER:
/PIPE-100 Extract PIPE/PIPE_EX7001
Q CLAIMLIST EXTRACT:
/PIPE-100
USERX1 creates PIPE-100
EXTRACT FLUSH DB PIPE/PIPE
Q CLAIMLIST:
none
Q CLAIMLIST OTHER:
none
Q CLAIMLIST EXTRACT:
/PIPE-100
USERX2:
EXTRACT REFRESH DB PIPE/PIPE
Q CLAIMLIST:
none
Q CLAIMLIST OTHER:
/PIPE-100 Extract PIPE/PIPE_EX7001
Q CLAIMLIST EXTRACT:
none
Note that USERX2 must use Q CLAIMLIST OTHER (not Q CLAIMLIST EXTRACT) to see the claim

Figure 6:1. Querying extract claimlists

When you create an element, Outfitting only sees it as a user claim, not an extract claim, until the element is flushed. It will then be reported as an extract claim (as well as a user claim, if it has not been unclaimed).

Note that a change in the claim status of an existing element will be shown by the appropriate Q CLAIMLIST command as soon as appropriate updates take place, but a user will have to GETWORK as usual to see the changes to the DESIGN model data.

We recommend that:

- Before you make a user or extract claim, you should do an EXTRACT REFRESH and GETWORK.
- If you need to claim many elements to an extract, it improves performance if the elements are claimed in a single command, for example, by using a collection:

```
EXTRACT CLAIM ALL FROM !COLL
```

Common Extract Commands

`Q DBNAME`	Returns the name of the database which you are actually writing to.
`Q CLAIMLIST`	Outputs a list of all elements currently claimed by yourself:
`Q CLAIMLIST OTHE`	Outputs a list of all elements currently claimed by other users who are accessing the same DB:
`Q CLAIMLIST EXTRACT`	Shows the extract claimlist for all the writable extracts in the MDB.
`Q CLAIMLIST EXTRACT DB dbname`	Shows the extract claimlist for the named extract DB.
`Q CLAIMLIST EXTRACT FREE DB dbname`	Shows the elements claimed to the current extract and **not** claimed to another extract or user. That is, the elements which can be **released**
`Q CLAIMLIST EXTRACT OTHER DB dbname`	Shows the elements claimed to the current extract **and** claimed to another extract or user.
`Q CLAIMLIST CONTROL DB dbname`	Shows the extract claimlist for a CONTROLLED named extract DB.
`Q DBAC`	Queries the access mode of the database. DBAC can have the text settings CONTROL, UPDATE or MULTIWRITE.
`Q DBCL`	Queries the claim mode of the database. DBCL can have the text settings EXPLICIT or IMPLICIT.
`Q LCLM`	Queries whether or not the current element is claimed by another user. Returns TRUE or FALSE.

Command Syntax:

```
>-- Q CLAIMLIST --+- OTHER -----.
                  |             |
                  |- EXTRACT ---+- OTHER --.
                  |             |          |
                  |             |- FREE ---|
                  |             |          |
                  |             '----------|
                  |                        |
                  |------------------------+-- DB dbname --.
                  |                                        |
                  '----------------------------------------+-->
```

6.1.4 Related Attributes

DAC related:

Attribute Name	Data Type	Qualifier	Description
DACCLA	BOOL		True if DAC allows element to be claimed
DACERR	STRING(120)	ATTR	Returns the DAC error
DACISS	BOOL		True if DAC allows element to be issued

Claim related:

Attribute Name	Data Type	Qualifier	Description
CLMID	STRING(120)		Unique system ID of user claiming element
CLMNUM	INTEGER		User or extract number claiming element. Extract numbers are negative
CLMTIE	ELEMENT(4)		Reference to elements that are automatically claimed along with this element
EXCLFR	BOOL		True if element claimed from this extract. Only True for Primary elements
EXCLHI	ELEMENT(5000)		Primary elements in descendant hierarchy claimed to this extract(includes this element)
EXCLTO	BOOL		True if element claimed to this extract. Only True for Primary elements
EXNCLH	ELEMENT(5000)		Primary elements in descendant hierarchy not claimed to this extract
EXTRC	STRING(120)		Name of extract claiming element
NPDESC	ELEMENT(5000)		List of non primary offspring
OKCLA	BOOL		True if element may be claimed
OKCLH	BOOL		True if element and hierarchy may be claimed
OKREL	BOOL		True if element may be released
OKRLH	BOOL		True if element and hierarchy may be released
PRIMTY	BOOL		True if element is primary
PRMMEM	BOOL		True if there are any primary elements amongst descendants
PRMOWN	ELEMENT		Primary owning element (will be itself if primary)
USCLHI	ELEMENT(5000)		Elements in descendant hierarchy claimed to this user
USERC	STRING(120)		User name of user claiming element
USNCLH	ELEMENT(5000)		Elements in descendant hierarchy not claimed to this user

Extract related:

Attribute Name	Data Type	Qualifier	Description
EXHCNC	ELEMENT(5000)		As EXTCNC, but repeat test for all descendents
EXHCNN	ELEMENT(5000)		As EXTCNN, but repeat test for all descendents
EXHCON	ELEMENT(5000)		As EXTCON, but repeat test for all descendents

Attribute Name	Data Type	Qualifier	Description
EXHRCN	ELEMENT(5000)		As EXRCN, but repeat test for all descendents
EXHRCO	ELEMENT(5000)		As EXTRCO, but repeat test for all descendents
EXMOC	BOOL		As EXMOD but ignoring changes to "noclaim" attributes and member lists
EXPMOC	BOOL		As EXPMOD but ignoring changes to "noclaim" attributes and member lists
EXPMOD	BOOL		True if primary and element or non-primary descendants have been modified in this extract
EXTCNC	ELEMENT(5000)		As EXTCON but excluding non modified elements
EXTCNN	ELEMENT(5000)		As EXTCON but excluding modified elements
EXTCON	ELEMENT(5000)		Primary elements connected/disconnected from element or non primary descendents in extract
EXTRCN	ELEMENT(5000)		As EXTCNN, but applied recursively to each connection
EXTRCO	ELEMENT(5000)		As EXTCON, but applied recursively to each connection
OKDROP	BOOL		True if element may be dropped
OKDRPH	ELEMENT(5000)		Primary elements preventing hierarchy drop
OKRLEH	ELEMENT(5000)		Primary elements preventing hierarchy release
OKRLEX	BOOL		True if element may be extract released

7 Undo and Redo

It is possible to undo and redo many operations. The undo mechanism is managed by Outfitting using a stack of transaction objects.

Each transaction object records the change in the state across the transaction.

The new descriptions are then:

MARKDB ‘comment’ - Complete the current transaction and starts a new transaction.

UNDODB - Undo the last transaction. If there is a current transaction then this is completed. Multiple Undos are allowed.

REDODB - Redo to next mark. Multiple Redos are allowed. A redo is only valid after an UNDO. Any database change after an UNDO invalidates a REDO.

7.1 How Undo Works

Every time you select an undo operation an entry is taken off the undo stack. The state saved in this transaction is restored, and the transaction object is placed on the redo stack.

When the undo stack is empty, then the **Undo** button and the **Undo** menu option will be greyed out indicating that there are no operations remaining that can be undone.

If the operation of undo involves moving into or out of model editing mode, then the switch into that mode will happen automatically, and the model editor button and menu option will reflect the change.

The selection set and handle appropriate to the editing operation that was being used will also be restored.

There are also a number of ways that you can perform an undo:

- By clicking on the **Undo** button on the appropriate toolbar.
- By selecting the **Undo** option on the **Edit** pulldown menu on the main toolbar.
- By entering the command `UNDODB n` where **n** indicates how many steps are to be undone.

The undo stack is automatically cleared after a SAVEWORK or GETWORK.

A similar process to the one described above occurs for redo.

When a transaction is taken off the redo stack, it is put back onto the undo stack.

If the user performs any operation that changes the database after doing an undo, then the redo stack will be cleared.

Refer to the *Software Customisation Guide* for controlling the undo stack from user defined PML.

MARKDB 'comment'	Set a Database mark. Multiple marks may be set.
UNDODB	Undo database to last mark. Multiple undos are allowed.
REDODB	Redo to next mark. Multiple Redos are allowed. A redo is only valid after an UNDO. Any database change after an UNDO invalidates a REDO.

The list of marks can be obtained from PML function MARKDB.

Example:

```
AREA 0
MARKDB 'First Mark'
AREA 100
MARKDB 'Second Mark'
AREA 200
MARKDB 'Third Mark'
AREA 300
!MARKS = MARKDB
Q VAR !MARKS
UNDODB
Q AREA - value will be 200
UNDODB
Q AREA - value will be 100
UNDODB
Q AREA - value will be 0
REDODB
Q AREA - value will be 100
REDODB
Q AREA - value will be 200
AREA 99
UNDODB
Q AREA - value will be 200
REDODB
Q AREA - value will be 99
```

The system will always create an initial mark the first time the database is changed.

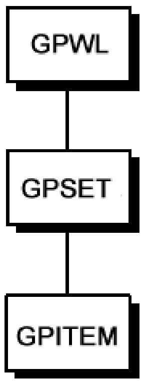
8 Groups & Secondary Hierarchies

A group element can hold in its members list a number of design elements from any combination of hierarchic levels, they may also span multiple DB's. You can use any appropriate design operation to act upon all of these individual elements simply by carrying out the operation on the group.

Groups are particularly useful when there is a need to create a secondary hierarchy of elements. For example a set of elements for a project may span more that one site, if this is the case it is difficulty to identify where in the hierarchy these elements occur. With a group you can easily query its members and see the hierarchy of elements contained within it.

A group is a DESIGN database element in its own right, and is therefore stored automatically for use in later sessions when you save database changes.

The Elements which make up a group within the DESIGN database are shown below:

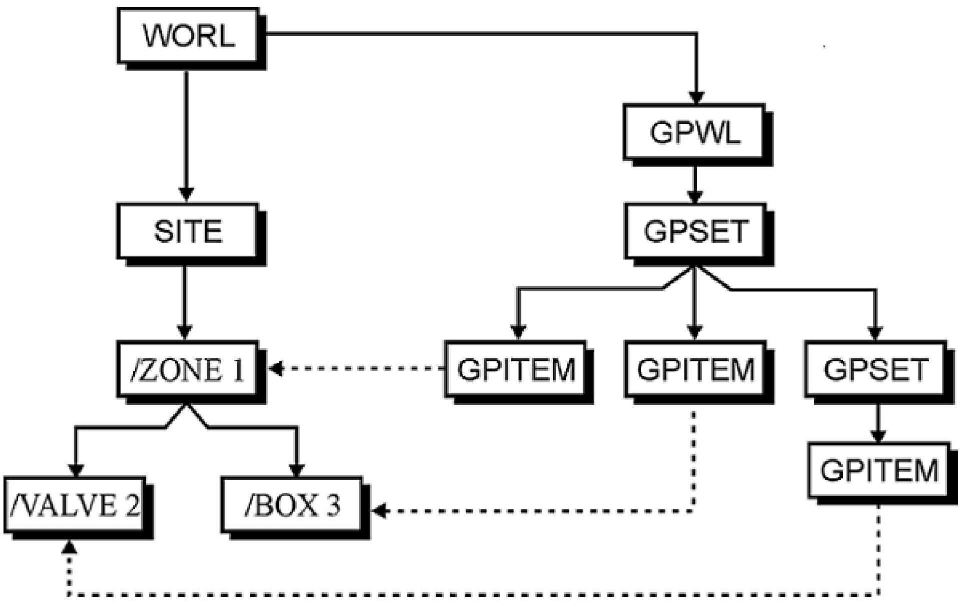


GPWL (Group World) Is a top level administrative element. A GPWL may hold multiple GPSET (Group Set) elements.

GPSET contains groups of items (GPITEM). A GPSET element has Name, DESC, and FUNCTION attributes.

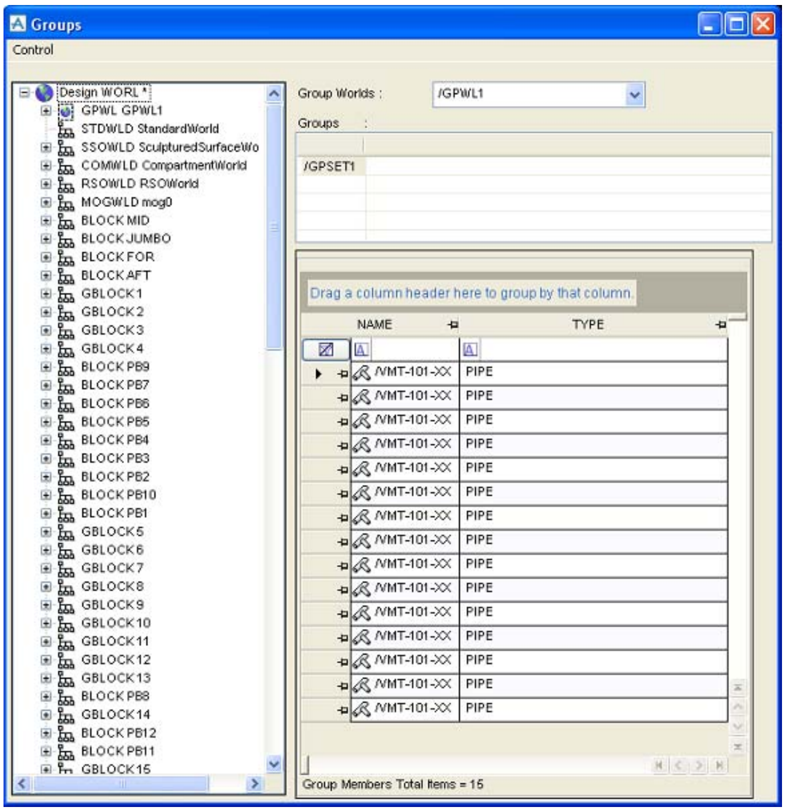
GPITEM These are elements within a database which are to be grouped under a Group Set (GPSET). Elements from different databases can all be grouped into the same Group Set. A GPITEM has the following attributes Name, DESC and SITEM.

It is possible to nest Group Sets within other Group Sets. To achieve this structure a GPSET can own another GPSET or a GPITEM can point back onto a GPSET. The following figure illustrates this:



Maintaining Groups using the Groups Form

The DESIGN module contains a user interface for maintaining and creating Groups, this is accessed through the Create > Group pulldown within the DESIGN Module. Selecting this will open the window below.



- Design WORL *
  - GPWL GPWL1
  - STDWLD StandardWorld
  - SSOWLD SculpturedSurfaceWo
  - COMWLD CompartmentWorld
  - RSOWLD RSOWorld
  - MOGWLD mog0
  - BLOCK MID
  - BLOCK JUMBO
  - BLOCK FOR
  - BLOCK AFT
  - GBLOCK1
  - GBLOCK2
  - GBLOCK3
  - GBLOCK4
  - BLOCK PB9
  - BLOCK PB7
  - BLOCK PB6
  - BLOCK PB5
  - BLOCK PB4
  - BLOCK PB3
  - BLOCK PB2
  - BLOCK PB10
  - BLOCK PB1
  - GBLOCK5
  - GBLOCK6
  - GBLOCK7
  - GBLOCK8
  - GBLOCK9
  - GBLOCK10
  - GBLOCK11
  - GBLOCK12
  - GBLOCK13
  - BLOCK PB8
  - GBLOCK14
  - BLOCK PB12
  - BLOCK PB11
  - GBLOCK15

Group Worlds : /GPWL1

Groups :

/GPSET1

Drag a column header here to group by that column.

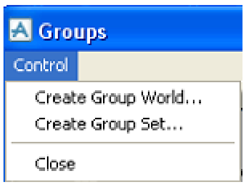
NAME	TYPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE

Group Members Total Items = 15

By default the form is populated with all GPWLs and GPSETs in the current MDB and defaults to the first GPSET in the first GPWL, the Group Members grid will be populated with the contents of the GPSET.

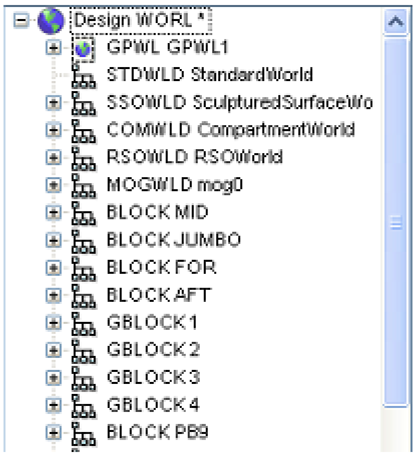
The Groups form has the following parts

Control Pulldown



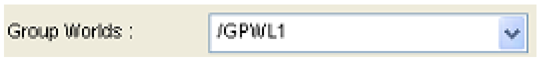
1. Create Group World - Allows the user to create a new Group World. A Group World is created in the hierarchy at the point of the currently selected item.
2. Create Group Set - Allows the user to create a new Group Set below the currently selected Group World.
3. Close - Will close the Groups form

Database Explorer Window



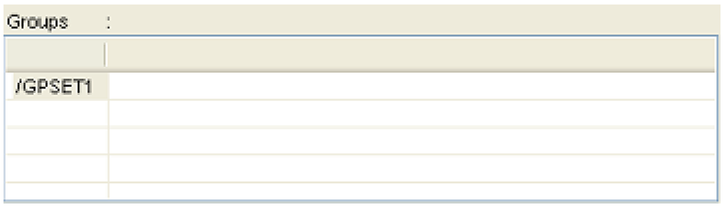
This allows the user to navigate the database hierarchy in exactly the same way as the standard explorer window in the DESIGN module. The benefit of having this accessible directly from this form allows the user to quickly select elements to include in a Group set.

Group Worlds Pulldown



This pulldown will expand to hold any Group Worlds created in the database hierarchy. Changing this pulldown will cause the Groups sub form to display all the Group Sets under the selected Group World.

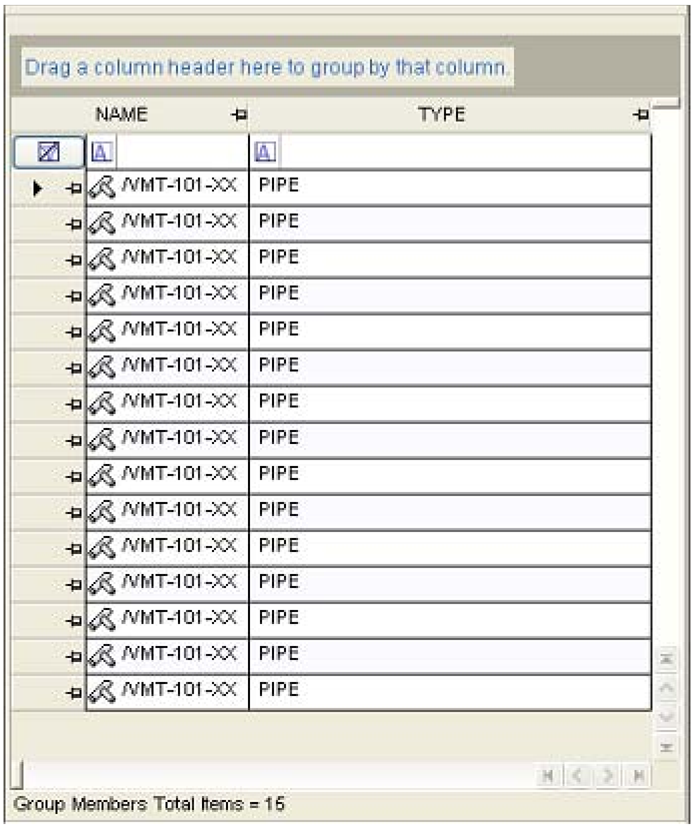
Groups Sub Form


Groups :

/GPSET1

The Groups sub form displays all the Group Sets which have been created below a Group World. Selecting a Group Set will cause the Group Members Grid to update.

Group Members Grid



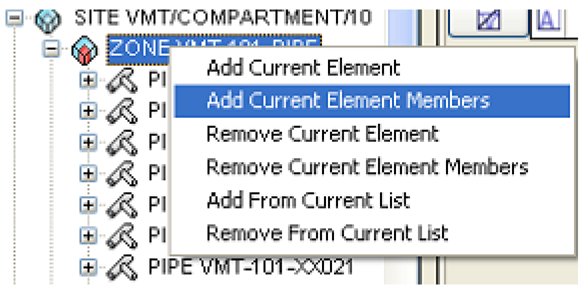
NAME	TYPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE
/VMT-101-XX	PIPE

Group Members Total Items = 15

This grid displays all of the elements which have been associated with the currently selected Group Set (from the Groups sub form).

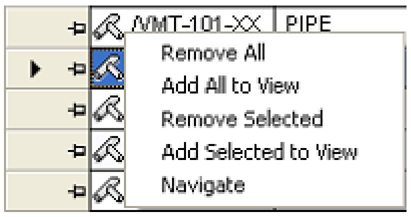
Maintenance of Groups is carried out through a set of menus accessible through right clicking the mouse.

Right clicking the mouse in the explorer window will give the following options



1. Add Current Element - Add the currently selected element to the Group Set selected in the Groups Sub Form
2. Add Current Element Members - Add currently selected element and all its members to the Group Set selected in the Groups Form
3. Remove Current Element - Remove the currently selected element from the Group Set selected in the Groups sub form
4. Remove Current Element Members - Remove the currently selected element and all its members from the Group Set selected in the Groups sub form
5. Add From Current List
6. Remove From Current List

Right Clicking the mouse in the Members Grid will give the following options.



1. Remove all - Remove all of the elements and their members from the Group Set currently selected in the Groups sub form
2. Add All to View - Will add all of the elements and members of the Group Set currently selected in the Groups sub form into the Design layout of the current project.
3. Remove Selected - Remove the currently selected element from the Group Set currently selected in the groups form.
4. Add Selected to View - Add the currently selected element into the design layout of the current project.
5. Navigate - Will move the explorer window to the position in the hierarchy of the selected element

Command line reference

Groups can be accessed through standard command line syntax. For example typing 'Q MEM' at a GPSET will return all the GPITEM elements associated with the Group.

The following sections summarise the primary methods of maintaining a Group, afterward are some examples of creating and querying Groups from the command line.

8.1 Defining Group Contents

The contents of a Group are defined by adding or removing references to or from the list part of the Group.

In order to use the commands described in this section, the current element *must* be the Group whose member list you wish to modify. Specified elements are then added to the list part of the current element starting from the current list position *or* are removed from the list part of the current element such that the current list position becomes the Head position.

The elements to be added to, or removed from, the Group's member list may be specified in any of the following ways:

- Explicitly, by name or (system-assigned) reference number.
- As members of specified elements, where a **member** of an element is defined as any element *immediately* below it in the DB hierarchy
- As items of specified elements, where an **item** of an element is any element anywhere below it in the hierarchy which has *no* list part (such as a Valve, Point, Box, etc.)
- By type (such as Equipment, Branch, Pipe, etc.)

Examples:

```
GADD /ZONE1 /VALVE2
```

Adds /ZONE1 and /VALVE2 to the current Group, starting from the current list position

```
GREMOVE /ZONE1 /BOX3
```

Removes /ZONE1 and /BOX3 from the current Group and moves the current list position pointer to the Head position

```
GADD MEM OF /BRANCH1 /BRANCH2
```

Adds all the pipe Components in Branches /BRANCH1 and /BRANCH2 to the current Group, starting from the current list position

```
GREM MEM OF /PIPE100 MEM OF /EQUI-B
```

Removes all Branches of the Pipe /PIPE100 and all members of Equipment /EQUI-B from the current Group

```
GREM ITEMS OF /ZONE2
```

Removes from the current Group all occurrences of those offspring of /ZONE2 which are items

```
GADD ALL EQU BRAN OF /ZONE1 /ZONE2
```

Adds all offspring of /ZONE1 and /ZONE2 which are of types Equip or Branch to the current Group, starting from the current list position

Command Syntax:

```
>--+-- GADD -----.     .-------------.
   |             |    /              |
   '-- GREMove --+---*-- <selatt> ---+--->
```

8.2 Deleting Groups

The action of this command differs from normal behaviour if the current element is a Group.

Examples:

`DELETE GPSET`

Only the current element and any Offspring that are GPSETs will be deleted.

`DELETE GPWLD`

Only the current element and any Offspring that are GPSETs will be deleted.

8.3 Copying a Group

Groups may be copied with a slightly different effect to normal elements.

Examples:

`COPY /GROUP21 (At a Group)`

The Current Group will contain exactly the same Members as /GROUP21. No new elements have been created.

9 Expressions

This section explains the PML 1 expressions package. These facilities are needed within AVEVA products, for example, to define report templates in Outfitting.

Note: Generally, all these facilities are compatible with PML 2.

Expressions have types. For example, you can have numeric expressions, text expressions and logical expressions. All the elements in an expression must be of the correct type. For example, if you have a two numbers, x and y, and two text strings text1 and text2, the following expression is meaningless:

```
x + text1              $
```

However, both of the following expressions are valid:

```
x + y                  $ adds the values of the numeric variables.
Text1 + text2          $ concatenates the two text strings.
```

The following types of expressions are available:

- *Logical Expressions*
- *Logical Array Expressions*
- *Numeric (Real) Expressions*
- *Numeric (Real) Functions*
- *Text Expressions*

9.1 Format of Expressions

The format of an expression, for example the use of brackets, spaces and quotes, is important. If you do not follow the rules given below you will get error messages:

Text must be enclosed in quotes. For example:

```
 'This is text'
```

There must be a space between each operator and operand. For example:

```
x + y
```

Use round brackets to control the order of evaluation of expressions and to enclose the argument of a function. For example:

```
SIN(30)
```

In general, you do not need spaces before or after brackets, except when an Outfitting name is followed by a bracket. If there is no space, the bracket will be read as part of the name. For example:

```
(NAME EQ /VESS1 )
```

9.1.1 Operator Precedence

Operators are evaluated in the order of the following list: the ones at the top of the list are evaluated first.

Operator	Comments
BRACKETS	Brackets can be used to control the order in which operators are evaluated, in the same way as in normal arithmetic
FUNCTIONS	
* /	
+ -	
EQ, NEQ, LT, LE, GE, GT	
NOT	
AND	
OR	

9.1.2 Nesting Expressions

Expressions can be nested using brackets. For example:

```
 ( (SIN(!angleA) * 2)  /  SIN(!angleB) )
```

9.2 Logical Expressions

Logical expressions can contain:

- Outfitting attributes of type logical e.g. BUILT.
- Logical constants. The constants available are: TRUE, ON, YES for true, and FALSE, OFF, NO for false.
- Logical operators.
- Logical functions.

9.2.1 Logical Operators

The logical operators available are:

Operator	Comments
AND	
EQ, NE	The operators EQ and NE may be applied to any pair of values of the same type.
GT, GE, LE, LT	The operators GE, LE, GT and LT may only be used with numbers and positions. For more information, see *Positions, Directions and Orientations in Expressions*.
NOT	
OR	

Note: The operators EQ, NE, LT, GT, LE and GE are sometimes referred to as ***comparator*** or ***relational*** operators; NOT, AND and OR are sometimes referred to as ***Boolean*** operators. See also *Precision of Comparisons* for tolerances in comparing numbers.

AND

Synopsis	`log1 AND log2                  -> logical`
Description	Perform the logical AND between two logical values. Treats unset values as FALSE.
Side Effects	If one of the values is undefined and the other one is FALSE, the result is FALSE.
Example	`TRUE and FALSE -> FALSE`

EQ and NE

Synopsis

```
( number1 EQual number2)     -> logical
( text1 EQual text2 )        -> logical
( log1 EQual log2 )          -> logical
( id1 EQual id2 )            -> logical
( pos1 EQual pos2 )          -> logical
( dir1 EQual dir2 )          -> logical
( ori1 EQual ori2 )          -> logical
( pp1 EQual pp2 )            -> logical
( number1 NEqual number2 )   -> logical
( text1 NEqual text2 )       -> logical
( log1 NEqual log2 )         -> logical
( id1 NEqual id2 )           -> logical
( pos1 NEqual pos2 )         -> logical
( dir1 NEqual dir2 )         -> logical
( ori1 NEqual ori2 )         -> logical
( pp1 NEqual pp2 )           -> logical
```

Description

Compare two values. A special feature is used for the positions, only the coordinates specified are compared. See *Comparing Positions* for more information. Unset values result in FALSE across EQ, TRUE across NE.

Side Effects

If two positions have no common coordinate, for example, `’N 10 ne U 10’`, the result is undefined. Units are consolidated across comparisons.

Example

```
( 1.0 eq 2.0) -> FALSE
```

Errors

None.

GT, GE, LE and LT

Synopsis	`( number1 GT number2 )        > logical`
	`( pos1 GT pos2 )              > logical`
	`( number1 GE number2 )        > logical`
	`( pos1 GE pos2 )              > logical`
	`( number1 LE number2 )        > logical`
	`( pos1 LE pos2 )              > logical`
	`( number1 LT number2 )        > logical`
	`( pos1 LT pos2 )              > logical`

Description	Compare two values. A special feature is used for positions: only the coordinates specified are compared. See *Comparing Positions* for more information. For positions, since comparisons may be performed on more than one value, LT (GT) is not the inverse of GE (LE). Unset values result in false
Side Effects	If two positions have no common coordinate, the result is undefined. For example, `'N 10 gt U 10`'. Units are consolidated across comparisons.

Example	`( 1.0 LT 2.0) -> TRUE`
	`( N 0 E 10 GT N 10 E 0 ) -> FALSE`
	`( N 0 E 10 GT N 10 E 0 )  -FALSE`

Errors	None.

NOT

Synopsis	`NOT log1                    -> logical`
Description	Perform the logical NOT on a logical value.
Side Effects	None.
Example	`not TRUE -> FALSE`
Errors	None.

OR

Synopsis	`OR log2                    -> logical`
Description	Perform the logical inclusive OR between two logical values. (The exclusive OR is defined by using NE.) Allows numbers instead of logical values.
Side Effects	If one of the values is undefined and the other one is TRUE, the result is TRUE.
Example	`TRUE or FALSE -> TRUE`
Errors	None.

9.2.2 Logical Functions

The logical functions available are:

Function	Comments
BADREF	
DEFINED,UNDEFINED	
CREATED	
DELETED	
EMPTY	
MATCHWILD	
MODIFIED	
UNSET	
VLOGICAL	

BADREF

Synopsis	`BADREF (id)                -> logical`
Description	TRUE if **id** is invalid, else FALSE.
Side Effects	None
Example	`BADREF(TREF)  ->  'true' if TREF=nulref`
Errors	None.

DEFINED and UNDEFINED

Synopsis	`DEFined (variable_name)      -> logical`
	`DEFined (variable_name,number)      -> logical`
	`UNDEFined (variable_name)      -> logical`
	`UNDEFined (variable_name , number)      -> logical`
Description	With one argument, DEFINED is true only if the scalar variable, the array variable or the array variable element exists.
	With two arguments, DEFINED is true only if the first argument is an array variable which has a value for the index denoted by the second argument.
	`UNDEFINED( !foo ) is equivalent to NOT DEFINED( !foo ).`
Side Effects	None.
Example	`DEFINED ( !var ) -> TRUE`
	`DEFINED ( !array ) -> TRUE`
	`DEFINED ( !array[1] )) -> TRUE`
	`DEFINED ( !array , 1 ) -> TRUE`
	`DEFINED ( !var) -> FALSE`
	`UNDEFINED ( !array) -> TRUE`
	`DEFINED ( !array , 3 ) -> FALSE`
Errors	None.

CREATED

Synopsis	`CREATED      -> logical`
Description	Returns TRUE if the element has been created since the set date.
Side Effects	None.
Example	`CREATED -> TRUE`
Errors	None.

DELETED

Synopsis	`DELETED                  -> logical`
Description	Returns TRUE if the element has been deleted since the set date.
Side Effects	None.
Example	`DELETED -> TRUE`
Errors	None.

EMPTY

Synopsis	`EMPTY(text)                -> logical`
Description	Returns TRUE if text is a zero length string, else FALSE
Side Effects	None.

Example	`EMPTY('') -> TRUE`
	`EMPTY('not empty') -> FALSE`

Errors	None.

MATCHWILD

Synopsis	`MATCHW/ILD( text1, text2)    -> logical`
	`MATCHW/ILD( text1, text2, text3)    -> logical`
	`MATCHW/ILD( text1, text2, text3, text4)    -> logical`

Description	Matches string **text2** to string **text1**. If they are the same then returns TRUE, else FALSE. **text2** may contain wildcard characters. The defaults for wildcards are '*' for any number of characters, and '?' for a single character. With three arguments, the multiple wildcard character '*' may be redefined by **text3**. With four arguments the single wildcard character '?' may be redefined by **text4**.
Side Effects	None

Example

```
MATCHW/ILD('A big bottle of
beer','*big*') -> TRUE
MATCHW/ILD('A big bottle of
beer','??big*') -> TRUE
MATCHW/ILD('A big bottle of
beer','???*big*') -> FALSE
MATCHW/ILD('A big bottle of
beer','*big*beer') -> TRUE
MATCHW/ILD('** text','**!','!') -> TRUE
```

Errors None.

MODIFIED

Synopsis

```
                                  .-----------------------------------.
                                 /                                    |
>- MODIFIED-(-+- attname -------*- DESCENDANTS --+-+-comma +-attname -'
              |                 |                | |
              |- DESCENDANTS -. |- SIGNIFICANT --| |
              |               | |                | |
              |- SIGNIFICANT--| |- PRIMARY ----- | |
              |               | |                | |
              |- PRIMARY -----| |- OFFSPRING-----| |
              |               | |                | |
              |- OFFSPRING ---| '----------------' |
              |               |                    |
              |               |                    |
              |               |                    |
              '---------------+--------------------+--+-- ) - OF - id →
                                                      |
                                                      '-→
```

Description

For sophisticated queries relating to modifications. Returns TRUE if a modification has taken place.

Each attribute name may be followed by the following qualifying keywords:

OFFSPRING, to check this element and members

SIGNIF, to check all elements for which this element represents the significant one;

PRIMARY, check all elements for which this element represents the primary one;

DESCENDANTS, this element and everything below (descendants).

The 'OF' syntax may be used as for attributes.

The MODIFIED function or the GEOM, CATTEXT and CATMOD pseudo-attributes.

The MODIFIED, DELETED and CREATED functions may go anywhere within an Outfitting PML1 expression. i.e. after Q/VAR and within collections

Side Effects None

Example

`Q MODIFIED()`	Returns TRUE if element has changed at all since the comparison date. It will also return TRUE if the element has been created since the comparison date.
`Q MODIFIED(POS,ORI)`	Returns TRUE if POS or ORI modified since the comparison date.
`Q MODIFIED(P1 POS)`	Returns TRUE if the position of P1 has changed.
`Q MODIFIED(GEOM DESCENDANTS`	Returns TRUE if any geometry for item or any descendants has changed
`Q MODIFIED(PRIMARY)`	Returns TRUE if any element for which this element is primary, has changed.
`Q MODIFIED() OF /PIPE1`	Returns TRUE if /PIPE1 has been modified since the comparison date.
`Q (BUIL OR MODIFIED()OR ELECREC OF NEXT )`	

Errors None.

The MODIFIED, DELETED and CREATED functions are not implemented within PML2 expressions.

UNSET

Synopsis `UNSET(value)` `-> logical`

Description Returns TRUE if **value** is unset, else FALSE. The value can be of any data type including ARRAYS. Normally it will be an Outfitting attribute.

Side Effects None.

Example	`UNSET( DESC )`	TRUE where **DESC** is an unset text attribute
	`UNSET(CRFA)`	FALSE where **CRFA** contains unset reference attributes

Errors None.

VLOGICAL

VLOGICAL is used for the late evaluation of variables.

Synopsis

```
VLOGICAL ( variable_name ))   -> logical
VLOGICAL ( variable_name ,    -> logical
number)
```

Description

With one argument, return the value of the scalar variable or the value of the array variable element as a logical.

With two arguments, return the value of the element corresponding to the index number as a logical.

The rules of conversion are:

TRUE for the strings ’T’, ’TR’, ’TRU’ or ’TRUE’ (case insensitive) or any numeric value not equal to zero;

FALSE for the strings ’F’, ’FA’, ’FAL’, ’FALS’ or ’FALSE’ (case insensitive) or a numeric value equal to zero.

Scalar variables may not be indexed. For example, `VTEXT(!var[1])` will return an error.

Array variables must have an index. For example, `VTEXT (!array)` will return an error.

The value cannot be translated into a logical.

See also VTEXT, used for late evaluation when a text result is required; and VVALUE, used for late evaluation when a numeric result is required.

Side Effects

If the scalar variable, the array variable, or the array variable element does not exist, the result is undefined.

Example

```
VLOG ( !array[1] )  -> TRUE
VLOG ( !array , 2 ) -> FALSE
```

Errors None.

9.2.3 Logical Array Expressions

Logical array expressions can contain:

- Outfitting attributes of type logical array. For example, LOGARR where LOGARR is a UDA of type logical.
- Logical constants. The constants available are: TRUE, ON, YES for true; and FALSE, OFF, NO for false.
- Logical operators. See *Logical Operators.*
- Logical functions. See *Logical Functions.*

9.3 Numeric (Real) Expressions

In Outfitting expressions, integers are treated as reals; they are fully interchangeable. Numeric expressions can contain:

- Numbers, for example: 32, 10.1.
- Numbers can be given as integer exponents, for example: 10 exp 5, and 5 E 6.
- Numbers can contain units. The valid units are MM, M/ETRES, IN/CHES, and FT, FEET. These may be preceded by SQU/ARE, CUBIC, CUB/E to denote non-linear values. For example: 100 mm, 10 exp 5 cubic feet. Feet and inches can be shown as, for example, 10'6:
- Outfitting attributes of type number, for example: XLEN.
- Position, direction and orientation attributes which have a subscript to indicate which part of the array is required. For example, `POS[2]` means the second element of the POSITION attribute; that is, the northing. Note that position, direction and orientation attributes without subscripts can only be used in number array expressions.
- The keyword PI (3.142).
- Numeric operators.
- Numeric functions.

9.3.1 Numeric (Real) Operators

The numeric operators available are:

Operator	**Comments**
+	Addition.
-	Subtraction.
*	Multiplication.
/	Division.

9.3.2 ADD and SUBTRACT (+ and -)"

Synopsis	`number + number`	`-> number`
	`number - number`	`-> number`
	`+ number`	`-> number`
	`- number`	`-> number`

Description	Add or subtract two numbers. They can also be used as unary operators at the beginning of a parenthesised sub-expression.
Side Effects	Units are consolidated across add and subtract.

Example	`1 + 2 -> 3.0`
	`1 - 2 -> 1.0`
	`+ 1 -> 1.0`
	`- 1 -> -1.0`

Errors	Floating point underflow.

9.3.3 MULTIPLY and DIVIDE (* and /)

Synopsis	`number * number            -> number`
	`number / number            -> number`

Description	Multiply or divide two numbers. They can also be used as unary operators at the beginning of a parenthesised sub-expression. Numeric underflow is not considered to be an error and neither is it flagged as a warning. The result returned is zero.
Side Effects	Units are consolidated across Multiply and Divide.

Example	`2 * 3 -> 6.0`
	`2 / 3 -> 0.666666666`

Errors	Divide by zero.

9.3.4 Numeric (Real) Functions

The numeric functions available are:

Function	Comments
`ABS ( number1 )`	Gives the absolute value of a number
`ACOS ( number1 )`	Gives the arc cosine of a number, in degrees.
`ASIN ( number1 )`	Gives the arc sine of a number, in degrees.
`ATAN ( number1 )`	Gives the arc tangent of a number, in degrees.
`ATANT ( number1, number2 )`	Gives the arc tangent of **number1/number2**, in degrees, with the appropriate sign.

Function	Comments
`ALOG ( number1 )`	Gives the exponential function (natural anti-log) of a number.
`ARRAY(pos` ***`or`*** `dir` ***`or`*** `ori)`	Converts a position, direction or orientation value or attribute into three numbers.
`ARRAYSIZE ( variable-name )`	Gives the size of an array variable.
`ARRAYWIDTH( variable-name )`	Gives the largest display width of any string in array variable-name.
`COMPONENT dir OF pos2`	Gives the magnitude of a vector drawn from E0 N0 U0 to `pos2`, projected in the direction **dir1**.
`INT ( number1 )`	Gives the truncated integer value of a number.
`SIN ( number1 )`	Gives the sine, cosine or tangent value of a number (considered to be in degrees).
`COS ( number1 )`	Gives the sine, cosine or tangent value of a number (considered to be in degrees).
`TAN ( number1 )`	Gives the sine, cosine or tangent value of a number (considered to be in degrees).
`LENGTH ( text1 )`	Gives the length of text1.
`DLENGTH ( text1 )`	Gives the length of **text1**. DLENGTH is used with characters which have a displayed width that is different from standard characters, such as Japanese.
`LOG ( number1 )`	Gives the natural logarithm of a number.
`MATCH ( text1, text2 )`	Gives the position of the beginning of the leftmost occurrence of **text2** in `text1`. If **text2** does not occur in **text1**, 0 is returned.
`DMATCH ( text1, text2 )`	Gives the position of the beginning of the leftmost occurrence of **text2** in `text1`. If **text2** does not occur in **text1**, 0 is returned. DMATCH is used with characters which have a displayed width that is different from standard characters, such as Japanese.
`MAX ( number1, number2[ , number3 [. . .]]) )`	Gives the maximum value of the arguments.

Function	Comments
`MIN ( number1, number2[ , number3 [. . .]]) )`	Gives the minimum value of the arguments.
`NEGATE`	Multiply a number by -1.0.
`NINT ( number1 )`	Gives the nearest integer to a real. NINT(N+0.5) is equal to N+1 if N is positive or equal to zero, to N if N is negative.
`OCCUR ( text1, text2 )`	Gives the number of times string **text2** occurs in string **text1**.
`REAL ( text1 )`	Try to read a number at the beginning of **text1**.
`POWER ( number1, number2 )`	Gives the value of **number1** raised to the power **number2**.
`SQRT ( number1 )`	Gives the square root of a number.
`VVALUE ( variable-name )`	Used for late evaluation of variables. Gives a real value.

ABS

Synopsis	`ABS ( number1 )` `-> number`
Description	Returns the absolute value of a real.
Side Effects	None.
Example	`ABS ( -3.2 ) -> 3.2`
Errors	None.

ACOS, ASIN, ATAN and ATANT

Synopsis

```
ASIN ( number1 )              -> number
ACOS ( number1 )              -> number
ATAN ( number1 )              -> number
ATANT ( number1, number2 )    -> number
```

Description	Return the arc-cosine, arc-sine or arc-tangent of a number, in degrees. ATANT returns the arc-tangent of `number1/number2` with the appropriate sign. ATANT is useful where the second value is near or equal to zero. For example, (6 0 ATANT) will give the correct result of 90 degrees, but (6 0 D ATAN) will indicate an error when trying to divide by zero.
Side Effects	None.
Example	`ACOS ( 0.8660254 ) -> 30`
Errors	Argument of ACOS or ASIN out of range [-1.0,+1.0] `ATANT (0.0,0.0)` is undefined.

ALOG

Synopsis	`ALOG ( number1 )              -> number`
Description	Return the exponential function (natural anti-log) of a number.
Side Effects	Numeric underflow causes the result to be set to zero.
Example	`ALOG( -0.7 ) -> 0.4965853`
Errors	Floating point overflow.

ARRAY

Synopsis	`ARRAY(pos` ***`or`*** `dir` ***`or`*** `ori)     -> number`
Description	Converts a position, direction or orientation value or attribute into three numbers.
Side Effects	None
Example	`ARRAY(e100 )  -> 100  0  0`
Errors	None.

ARRAYSIZE

Synopsis	`ARRAYSize ( variable-name )    -> number`
Description	Give the size of an array variable.

Side Effects	If the array variable does not exist, the result is undefined.
Example	`ARRAYSIZE(!array)  -> 2.0`
Errors	The variable is a scalar variable and not an array variable. The variable is an array variable element and not an array variable.

ARRAYWIDTH

Synopsis	`ARRAYWIDTH ( variable-name )  -> number`
Description	Give the largest display with of any string in array **variable_name**.
Side Effects	None.
Example	If an array contains the following values:

```
!ARRAY[1]      'Bread'
!ARRAY[2]      'is'
!ARRAY[3]      'for'
!ARRAY[4]      'life,'
!ARRAY[5]      'not'
!ARRAY[6]      'just'
!ARRAY[7]      'for'
!ARRAY[8]      'breakfast'
```

Then

```
ARRAYWIDTH(!ARRAY -> 9
```

i.e. the length of 'breakfast'.

Errors	The variable is a scalar variable and not an array variable. The variable is an array variable element and not an array variable.

COMPONENT ... OF ...

Synopsis	`COMPonent dir1 OF pos2      -> text`
Description	Returns the magnitude of a vector drawn from E0 N0 U0 to **pos2**, projected in the direction **dir1.**
Side Effects	None.
Example	`COMP E 45 N of N 0 E 100 U 50 -> 70.710`
Errors	None.

SINE, COSINE and TANGENT

Synopsis	`SINe ( number1 )          -> number`
	`COSine ( number1 )        -> number`
	`TANgent ( number1 )       -> number`

Description	Return the sine, cosine or tangent value of a number (considered to be in degrees).
Side Effects	None.

Example	`COS ( 0.0 )  -> 1.0`
	`TAN ( 45.0 )  -> 1.0`

Errors	Division by zero for TAN if the sine is (nearly) equal to zero.

INT

Synopsis	`INT ( number1 )           -> number`
Description	Return the truncated integer value of a number.
Side Effects	None.

Example	`INT ( 1.6 )    -> 1.0`
	`INT ( -23.7 )   -> -23.0`

Errors	Integer overflow.

LENGTH and DLENGTH

Synopsis	`LENgth ( text1 )          -> number`
	`DLENgth ( text1 )         -> number`

Description	Return the length of **text1**. DLENGTH is for use with characters which have a displayed width that is different from standard characters, such as Japanese.
Side Effects	None.

Example	`LENGTH ( 'abcdef' )  -> 6.0`
	`LENGTH ( '' ) -> 0.0`

Errors	None.

ALOG

Synopsis	`LOG ( number1 )              -> number`
Description	Return the natural logarithm of a number..
Side Effects	None.
Example	`LOG( 3 ) -> 1 0986123`
Errors	Negative arguments.

MATCH and DMATCH

Synopsis	`MATch ( text1 , text2)       -> number`
	`DMATch ( text1 , text2)      -> number`

Description	Return the position of the beginning of the leftmost occurrence of **text2** in **text1**. If **text2** does not occur in **text1**, 0 is returned DMATCH is for use with characters which have a displayed width that is different from standard characters, such as Japanese.
Side Effects	None.

Example	`MATCH ( ’abcdef’ , ’cd’ ) -> 3.0`
	`MATCH ( ’abcdef’ , ’x’ ) -> 0.0`
	`MATCH ( ’abcdef’ , ’’ ) -> 1.0`

Errors	None.

MAX and MIN

Synopsis	`MAX ( number1 , number2 [ , number3 [ ... ] ] )   -> number`
	`MIN ( number1 , number2 [ , number3 [ ... ] ] )   -> number`

Description	Return the maximum or minimum value of the arguments.
Side Effects	None.

Example	`MAX ( 1 , 3.4 )         -> 3.4`
	`MIN ( 7.6 , -12.33 , 2.3 )  -> -12.33`

Errors	None.

NEGATE

Synopsis	`NEGate ( number1 )            -> number`
Description	Multiply a real by -1.0.
Side Effects	None.
Example	`NEG ( 1 ) -> -1.0`
Errors	None.

NINT

Synopsis	`NINT ( number1 )              -> number`
Description	Return the nearest integer to a real. `NINT(N+0.5)` is equal to `N+1` if **N** is positive or equal to zero, to **N** if **N** is negative.
Side Effects	None.

Example	`NINT ( 1.1 )   -> 1.0`
	`NINT ( -23.7 )  -> -24.0`
	`NINT ( 1.5 )   -> 2.0`
	`NINT ( -11.5 )  -> -12.0`

Errors	Integer overflow.

OCCUR

Synopsis	OCCUR(text1, text2) `-> integer`
Description	Counts the number of times string **text2** occurs in string **text1**
Side Effects	None.

Example	`OCCUR ('ABBACCBBBBBAB', 'BB')  -> 3`
	`OCCUR('ZZZZZZZZZZZ', 'A')  -> 0`

Errors	None..

REAL

Synopsis	`REAL ( text1 )                -> number`
Description	Try to read a real number at the beginning of **text1**. Note that if text is in the form of an exponent, (-12E-1 in the third example), there must be no spaces in it. Note: this function was formerly called NUMBER.
Side Effects	Numeric underflow causes the result to be set to zero. Units are consolidated across POWER.

Example	`REAL ( ’12.34’) -> 12.34`
	 `REAL ( ’ 7.23 E 3 meters’ )  -> 7.23`
	 `REAL ( ’ -12E-1 meters ’ )  -> -1.2`

Errors	Unable to convert the text into a real number.

POWER

Synopsis	`POWer ( number1 , number2 )  -> real`
Description	Return the value of **number1** raised to the power **number2**.
Side Effects	None.
Example	`POWER ( -2 , 3 ) -> -8`
Errors	Floating point overflow. Zero first argument and non-positive second argument (effectively divide by zero). Negative first argument and non-integer second argument.

SQRT

Synopsis	`SQrt ( number1 )              -> number`
Description	Return the square root of a real.
Side Effects	Units are consolidated across SQRT.
Example	`SQRT ( 4 ) -> 2.0`
Errors	Negative argument.

VVALUE

VVALUE is used for the late evaluation of variables.

Synopsis	`VVALue( variable_name )        -> number`
	`VVALue( variable_name , number )        -> number`

Description	With one argument, returns value of the scalar variable or value of the array variable element as a number. With two arguments, returns value of the element corresponding to the index number as a number. See also VLOGICAL, used for late evaluation when a logical result is required, and VTEXT, used for late evaluation when a text result is required.
Side Effects	If the scalar variable, the array variable or the array variable element does not exist, the result is undefined.

Example	`VVAL ( !array[1] ) -> 1.0`
	`VVAL ( !array , 2 ) -> 0.0`

Errors	Scalar variable may not be indexed. For example, VTEXT `(!var[1]) )` will return an error. Array variable must have an index. For example, VTEXT `( !array ) )` will return an error. The string can not be converted to a number.

9.3.5 Real Arrays

Real array expressions can contain attributes of type real array, for example: DESP.

9.4 Using IDs in Expressions

IDs can be used in expressions. IDs can be any of the following:

- Element name, for example: /VESS1.
- Refno, for example: =23/456.
- Element type further up the hierarchy, for example: SITE.
- Number within member list, for example: 3.
- Type and number within member list, for example: BOX 3.
- NEXT, PREV for next, previous within current list. Optionally with a count and/or element type, for example: NEXT 2 BOX, LAST CYL.
- NEXT, PREV MEMBER for next, previous within member list. Optionally with a count and/or element type.
- If the element type given is only valid as a member then MEMBER is assumed. For example, NEXT BOX at an EQUIPMENT will assume MEMBER.

- FIRST, LAST for first and last in current list. Optionally with a count and/or element type.
- FIRST, LAST MEMBER for first and last in member list. If the element type given is only valid as a member then MEMBER is assumed.
- END to navigate up from current list.
- END is similar to owner but not quite the same. For example, if the current element is a GROUP MEMBER, and it has been reached from the GROUP then END will return to the group but OWNE will go to the true owner.
- Attribute of type ref, for example: CREF
- SAME to mean last current element
- NULREF to mean =0/0
- CE for the current element
- 'OF' may be used to nest the options indefinitely. For example:

```
SPEC OF SPREF OF FLAN 1 OF NEXT BRAN.
```

- This denotes the SPEC element owing the SELE element pointed to by the SPREF attribute on the first FLANGE of the next BRANCH. ILEAVE TUBE, IARRIV TUBE, HEAD TUBE, TAIL TUBE can be added to denote tube. For example:

```
HEAD TUBE OF /BRAN1.
```

- An error will occur if there is no implied tube for the element concerned.

  ID arrays can also be used in expressions. For example, CRFA.

Note: Some of the ID syntax clashes with other types. To allow for this, an id expression may always be preceded with the keyword ID. For example, ID 3 will mean the third member of the current list rather than a number of value 3.

9.5 Positions, Directions and Orientations in Expressions

9.5.1 Using Positions in Expressions

The basic ways of defining a position are:

- Position attribute plus optional WRT. For example:

```
POS OF /VESS1 WRT /* or P1 POS OF /CYL2
```

- Cartesian position. For example:

```
N 45 W 20000 U 1000
```

- Cartesian position from an element. For example:

```
N 1000 FROM /ATEST.
```

- Cartesian position from a ppoint. For example:

```
N 1000 FROM P1 OF /BOX2.
```

- Cartesian position from an attribute. For example:

```
N 1000 FROM POSS OF /SCTN1
```

- Any numeric value within a position may itself be an expression. For example: the following is a valid position expression

```
N (DESP[1] + 10) E
```

The Cartesian position may optionally be followed by WRT to specify the axis system. See *WRT*.

9.5.2 WRT

The WRT keyword is used to toggle between absolute and relative units.

When we specify an element (or attribute of an element) we are specifying an absolute point in world space. The point can be given in world space or some other axis. Normally the answer is required relative to the owner axis system and this is taken as the default. For example:

`Q POS`	$ will return the position of the current element
	$ relatively to its owner.
`Q POS OF /EQUIP1`	$ will return the position of EQUIP1 relative to its
	$ owner.

If we require the result in some other axis system then the WRT keyword is used. For example:

`Q POS WRT /*`	$.for the position in world coordinates.

When we specify a Cartesian coordinate we are dealing with a relative position.

For example, ’N 10’ is meaningless until we specify the axis system, or default to an axis system.

Again we use WRT to do this, although it is important to note that in this case we are going from a relative position to an absolute position (in the previous example WRT was used to go from an absolute position to a relative one).

For example:

`N 100 WRT /BOX1`	$ specifies an absolute position in world space
	$ which is N100 of /BOX1.

The default is that Cartesian coordinates are in the owning element’s axis system. This absolute position can be expressed in different coordinate systems: the default is again the owner’s axis system.

Note: The CONSTRUCT syntax uses the world as the default axis

Example

Item	**Comments**
A SITE at (0,0,0)	With default (World) orientation
A ZONE at (100,0,0)	With default (World) orientation

Item	Comments
An EQUIPMENT at (100,0,0)	With orientation ’N IS E
A BOX at (-100,0,0)	With default (World) orientation

The result of `Q (N 100 WRT /BOX1)`, will depend on the current element.

Location	Result
World	(300,100,0), in World coordinates.
Site	(300,100,0) in World coordinates because the World is the owner of the current element.
Zone	(300,100,0) in World coordinates, because the Site is the owner of the current element, and the Site coordinates are the same as the World coordinates.
Equipment	(200,100,0), which is the position relative to its owner, the Zone.
Box	(100,100,0) which is the position relative to its owner, the Equipment.

WRT can be further qualified by FROM.

9.5.3 FROM

In some cases we require an offset from a fixed point, other than the position of an item. For example, a point or attribute.

The FROM syntax is used for this. We may still use WRT in combination with FROM, but in this case the WRT is only used to determine the axis direction and not the offset, since the offset is specified by the FROM part.

Consider the following:

Item	Comments
A SITE at (0,0,0)	With default (World) orientation
A ZONE at (100,0,0)	With default (World) orientation
An EQUIPMENT at (100,0,0)	With orientation ’N IS E
A BOX at (-100,0,0)	With default (World) orientation

The result of `Q (N 100 WRT /* FROM /BOX1)`, shown as ⊗ in , will depend on the current element.

Location	Result
World, Site, and Zone	(200,200,0) since the offset of N100 is applied in world axis rather than /BOX1 axis.
Equipment	(100,200,0). Note: the default axis for the result is the Zone.
Box	(200,0,0), because the default axis for the result is the Equipment.

The result of `'Q (N 100 WRT /BOX1 FROM /* )` is different:

Location	Result
Site and Zone	(100,0,0)
Equipment	(0,0,0)
Box	(0, -100, 0), because the axis for the result is the Equipment.

The result of `'Q (N 100 FROM /* )'` is different yet again.

For this we cannot mark an absolute point on the diagram since the default WRT will vary with the current element. In fact for the SITE, ZONE, EQUI the point ⊗ is marked in , and for the BOX the point coincides with the ZONE.

Location	Result
Site and Zone	(0,100,0)
Equipment	(-100,100,0), because the default result axis is the Zone.
Box	(0, -100, 0), because the axis for the result is the Equipment.

9.5.4 Comparing Positions

Two positions can be compared with EQ, NE, GT, LT, GE or LE. The pairs of coordinates are only compared in the coordinate axes for which the two positions are defined. A position attribute always has all three coordinates defined.

For positions entered by the user, only those coordinates which are given by the user are defined. For example:

`'N10U3'`	$ only the Y and Z coordinates are defined,
	$ while the X coordinate remains undefined

For the EQ operator, all the pairs of defined coordinates should be equal. For NE, only one pair of defined coordinates need be different. For GT (LT,GE,LE), all the defined coordinates of the first position should be greater than (less than, greater than or equal to, less than or equal to) the defined coordinates of the second position. This means that GE is not the opposite of LT and LE is not the opposite of GT.

If no coordinate of the two positions are defined for a common axis (e.g. 'N10' and 'W4D7'), the result of the comparison is undefined.

Examples

`'POS EQ W1S2D3'`	$ This evaluates to true only if POS of the current $ element is (-1,-2,-3).
`'POS GT N10' or 'N10 LE POS'`	$ Only the second coordinate of POS is compared; $ if it is greater than 10, then the result is true.

`'E10N10 GT E0N0'`	$ Is true because the inequality is verified for the X
	$ and Y axis (both coordinates are undefined for
	$ the Z axis, so it is ignored).

`'E10N0 GT E0N0'`	$ Is false because the Y components are different $ axes.

`'E10N0 GT E0U100'`	$ Is true. Although no comparison can be
	$ performed n either the Y or the Z axis, because
	$ the components are not present in both position
	$ constants, the comparison is true in the X
	$ component.

`'N10 EQ W4D7'`	$ Is undefined (no comparison is possible).

See also *Precision of Comparisons*, for tolerances in comparing numbers.

9.5.5 POLAR

The POLAR keyword allows positions to be defined in terms of a distance in a particular direction from a point.

The syntax is:

```
POLAR dir DISTance expr -+- FROM -+- pos -----.
                         |        |           |
                         |        '- point ---|
                         |                    |
                         '--------------------+--->
```

If FROM is not specified the default is the origin of the owner.

For example:

```
POLAR N 45 E DIST 20M FROM U 10 M

POLAR AXES PL OF PREV DIST ( ABORE * 10 ) FR
OM PL OF PRE V
```

9.5.6 Direction

The basic ways of defining a direction are:

- Direction attribute plus optional WRT. For example,

  `HDIR OF /PIPE1 WRT /*`

- Cartesian direction. For example,

  `N 45 W`

- Cartesian direction WRT to an element.
- All Cartesian directions are returned in the axis of the owner of the current element. For example:

  `(U WRT CE )`

- will return the Z axis of the current element relative to its owner.

  `Q ( Z WRT /SCTN )`

- will return the Z axis direction of /SCTN relative to the owner of the current element. For example, if the result is required in world coordinates the current element must be the World or a Site.
- FROM pos2 TO pos2. For example

  `FROM N 50 WRT CE TO N 100`

- Keyword AXES followed by a p-point or pline.
- The CLOSEST keyword, which will find the closest element in a particular direction. The syntax is:

```
>- CLOSEST type -+- WITH exp -.
                 |           |
                 ‘-----------+- DIRECTION dir -+- EXTENT val -.
                                               |              |
                                               ‘--------------+--> cont

         continued >-+- AFTER val -.
                     |             |
                     ‘-------------+- FROM ? -.
                                   |          |
                                   ‘----------+-->
```

- In the above graph the keywords are:
- EXTENT, which is how far to search in the direction specified, default 10M
- AFTER, or the distance along vector after which to start search, default 0M
- FROM, which specifies an alternative start point other than current element. This is of particular use for a branch where you might want to specify the HPOS or TPOS.
- Examples are:

```
CLOSEST DIR E
```

```
CLOSEST BOX WITH ( PURP EQ 'FLOO' ) DIR D WRT /
* EXTENT 20M
```

```
CLOSEST VALVE DIR N 45 U FROM E100 N200 U300
```

```
CLOSEST BRAN HANG AFTER 2M
```

9.5.7 Orientations

The basic ways of defining an orientation are:

- Orientation attribute plus optional WRT. For example:

  ```
  ORI OF /BOX1 WRT /*
  ```

- Cartesian orientation. For example:

  ```
  dir IS dir AND dir IS dir
  ```

- For example to set an orientation of an element to that of a section, rotated by 90 degrees use:

  ```
  (E IS U WRT /SCTN1 AND N IS E WRT /SCTN1)
  ```

- The AXES keyword, which will allow you to use P-points to specify orientations.
- The syntax is:

```
                 ----<---------.
                /              |
>-- AXES --*--- PArrive ---|
               |               |
               |--- PLeave ----|
               |               |
               |--- PTail -----|
               |               |
               |--- HHead -----|
               |               |
               |--- HTail -----|
               |               |
               '--- PPOINT n --+-- OF - <gid> ---->
```

- An example is:

  ```
  ( AXES PLEAVE IS AXES PLEAVE OF PREV AND AXES P3 IS UP )
  ```

- This will orient a branch component, such as a valve, so that it is aligned with the previous component and its P3 is up.

  See also *Comparing Positions*.

9.6 Text Expressions

Text expressions can contain the following:

- A text string, which must be enclosed in quotes. For example: 'FRED'.
- An Outfitting attribute of type text or word. For example: FUNC
- A single element of a word array attribute. For example: ELEL[2].

- Text operators
- Text functions

9.6.1 Text Operator

The text operator available is +, used for concatenation.

Synopsis	`text1 + text2 -> text        -> text`
Description	Return the concatenation of two text strings.
Side Effects	None.
Example	`’no’ + ’space’ -> ’nospace’`
Errors	Text result too long.

9.6.2 Text Functions

The text functions available are:

Function	Comments
AFTER	
BEFORE	
DISTANCE	
LOWCASE, UPCASE	
PART	
REPLACE	
STRING	
SUBS, DSUBS	
TRIM	
VTEXT	

AFTER

Synopsis	`AFTER ( text1 , text2 )      -> text`
Description	Return the substring of **text1** which is after the leftmost occurrence of **text2** in **text1**. If **text2** does not occur in **text1**, the null string is returned.
Side Effects	None.

Example

```
AFTER ( ’abcdef’ , ’cd’ )  ->’ef’
AFTER ( ’abcdef’ , ’x’ )  -> ’’
AFTER ( ’abcdef’ , ’’ )   -> ’abcdef’
```

Errors None.

BEFORE

Synopsis

```
BEFORE ( text1 , text2 )    -> text
```

Description Return the substring of **text1** which is before the leftmost occurrence of **text2** in **text1**. If **text2** does not occur in **text1**, text1 is returned.

Side Effects None.

Example

```
BEFORE ( ’abcdef’ , ’cd’ ) -> ’ab’
BEFORE ( ’abcdef’ , ’x’ )  -> ’’
BEFORE ( ’abcdef’ , ’’ )  -> ‘’
```

Errors None.

DISTANCE

Synopsis

```
DISTance ( number1 )      -> text

DISTance( number1,        -> text
logical1, logical2,
logical3, number2,
logical4)
```

Description

For the one-argument form, if the current distance units are FINCH, text is the conversion of the decimal inches value **number1** into the format 'aa'bb.cc/dd'. Otherwise, text is the STRING conversion of **number1**.

The six-argument form is more complex. The format is:

```
DIST/ANCE (distance, feet, usformat,
fraction, denom_or_dp, zeros)
```

where:

- **distance** is the numeric distance in inches that is to be formatted.
- **feet** is a logical flag set to true if output is to be in feet and inches and to false if output is to be in inches.
- **usformat** is a logical set to true if US format is to be used or false if Outfitting format is to be used.
- **fraction** is a logical set to true if the fractional component is to be output as a fraction or false if to be output as a decimal denom_or_dp is a number representing the largest denominator if **fraction** is TRUE or representing the number of decimal places if it is FALSE.
- **zeros** is a logical set to true if zeros are to be shown when that component of the output has no value

Outfitting

For both US and Outfitting formats the following rules are observed:

- If **distance** is negative, the first symbol is a minus sign.
- If **feet** is true and the distance is at least a foot, then the number of feet is output next, followed by a single quote ('). Only if zeros is true will the number of feet be output as 0 for distances less than a foot. Otherwise the feet will be omitted.
- If feet have been output, the inches will be at least two characters wide. Numbers less than ten will be preceded by a space if US format is being used or a zero if Outfitting format is used. A zero will be output if there are no whole inches.
- If no feet have been output and the distance is at least an inch, then the number of inches is displayed but without any preceding spaces. Only if zeros is true will a 0 be output for distances of less than an inch.
- If inches have been output and **fraction** is true, these will be followed by a decimal point (.).
- If **fraction** is TRUE and the number has a fractional component, then the numerator and the denominator are shown separated by a slash (/). This is then blank padded up to the width that the largest numerator and denominator would take.

- If fraction is FALSE and the number of decimal places is greater than zero, then the decimal point (.) is displayed followed by the remainder up to the appropriate number of decimal places. If the number of decimal places is 0 then the decimal point is not shown either.
- If US format has been selected then the following additional rules are observed on output:
- The (’) after the number of feet is followed by a dash (-).
- The decimal point separating the inches from the fraction is replaced by a space.
- The inches and fraction of inches are followed by a double quote(”).

Side Effects None.

Example If the current distance units are FINCH:

```
DISTANCE ( 17.5 ) ->  ’1’5.1/2’
```

Some examples, where the current distance units are feet and inches:

```
DIST(34.5,TRUE,TRUE,TRUE,100,TRUE) -> 2’-10.1/2.
DIST(34.5,FALSE,TRUE,FALSE,1,TRUE) -> 34.5”
DIST(34.5,FALSE,TRUE,TRUE,4,FALSE) -> 34 1/2”
DIST(128.5,TRUE,FALSE,TRUE,2,TRUE) -> 10’08.1/2”
```

The following table shows sets of options that could have been chosen and the format of the output produced for different numbers. Blanks output by the system are represented by underscores(_).

Distance	Feet & Inch US Fraction Denom 100 Zeros	Feet & Inch US Fraction Denom 32 No Zeros	Inches US Decimal DP 1 Zeros	Inches US Fraction Denom 4 No Zeros	Feet & Inch Outfitting Fraction Denom 2 Zeros
128.5	10’-_8_1/2”___	10’-_8_1/2”__	128.5”	128_1/2”	10’08.1/2
120.0	10’-_0”_______	10’-_0”______	120.0”	120”____	10’00____
11.5	0’-11_1/2”___	11_1/2”__	11.5”	11_1/2”	0’11.1/2
0.75	0’-_0_3/4”___	3/4”__	0.8”	3/4”	0’01____
0.0	0’-_0”_______	______	0.0”	____	0’00____
-10.0	-0’-10”_______	-10”______	-10.0”	-10”____	-0’10____

Errors The value is too big to be converted.

LOWCASE and UPCASE

Synopsis

```
UPCase ( text1 )            -> text

LOWCase ( text1 )           -> text
```

Description

Return an upper or lower case version of **text1**.

Side Effects

None.

Example

```
UPCASE ( 'False') -> 'FALSE'
LOWCASE ( 'False') -> 'false'
```

Errors

None.

PART

Synopsis

```
PART(text1, number1)        -> text

PART(text1, number1 ,       -> text
text2)
```

Description

With two arguments, returns the `number1` component of `text1` assuming that `text1` is split on any whitespace characters. If `number1` is negative, counting of components starts from the right.

With three arguments, as above, but use `text2` as the separator on which splitting takes place.

If the user gives a part number higher than the number of components in the string, the function returns an empty string.

Side Effects

None.

Example

```
PART ('x-y-z', 1, '-' -> 'x'
PART ('a    b   c            d e', 4-> 'd'
PART ('/PIPE45/B9', -1, '/')  -> 'B9'
PART('aa bb cc', 2) ->  'bb'
PART('aa-bb-cc',3,'-')  -> 'cc'
```

Errors

None.

REPLACE

Synopsis

```
REPLace (text1,text2,text3)  -> text
```

```
REPLace(text1,text2,text3,i  -> text
nt1)
```

```
REPLace(text1,text2,text2,i  -> text
nt1,int2)
```

Description

Replace search string **text2** in input string **text1** with replacement string **text3**.

If **int1** is given this specifies the first occurrence of **text2** at which to start replacement.

If **int2** is given this specifies the number of replacements to make. int1 and/or **int2** may be negative to indicate that the direction is backwards.

Side Effects

None.

Example

Three arguments:

```
REPLACE ('cat dog cat cat dog ', 'cat',
'dog' ) -> 'dog dog dog dog dog'
```

All occurrences of 'cat' are replaced with 'dog'.

Four arguments: start occurrence given:

```
REPLACE ('cat dog cat cat cat dog', 'cat',
'dog', 2) -> 'cat dog dog dog dog dog
```

All occurrence of 'cat' from the second occurrence onwards are replaced with 'dog'

```
REPLACE('cat dog cat cat dog' ,'cat',
dog', -2 -> 'dog dog dog cat dog'
```

All occurrences starting at the second occurrence from the end of the string and moving backwards are replaced Note that a negative fourth argument without a fifth argument implies backwards mode.

Five arguments: start occurrence and number of replacements given. Replace two occurrences of 'cat' starting at second occurrence:

```
REPLACE ('cat dog cat cat cat, 'cat',
'dog', 2,2) -> 'cat dog dog dog cat'
```

Replace two occurrences in backwards direction starting at the second occurrence:

```
REPLACE ('cat dog cat cat cat', ,'cat',
'dog', 2, -2) -> 'dog dog dog cat cat '
```

Replace two occurrences in forwards direction starting at second occurrence from the end of the string:

```
REPLACE ('cat cat cat cat dog', 'cat',
'dog',-2,2) -> 'cat cat dog dog dog'
```

Replace two occurrences in backwards direction starting at second occurrence from the end of the string.

```
REPLACE ('cat cat cat cat dog','cat',
'dog', -2, -2) -> 'cat dog dog cat dog'
```

The following examples all give the same result:

```
REPLACE('cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8
cat9 cat10', 'cat', 'dog', 4, 2)
REPLACE('cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8
cat9 cat10', 'cat', 'dog', 5, -2)
REPLACE('cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8
cat9 cat10', 'cat', 'dog',-6, -2)
REPLACE('cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8
cat9 cat10', 'cat', 'dog', -7, 2)
```

In each case, the output string is

```
'cat1 cat2 cat3 dog4 dog5 cat6 cat7 cat8
cat9 cat10'
```

If the replacement string **text3** is a null string the required number of occurrences of the search string **text2** are removed. For example:

```
REPLACE ('AAABBABZ', 'B', '') ->     'AAAAZ'
REPLACE ('AAABBABZ', 'B', '', -1, -1) ->
'AAABBAZ'
```

Errors

If the input string **text1** is a null string or an unset text attribute, the input string **text1** is returned unchanged. For example:

```
REPLACE ('', 'A','B')  ->  ''
```

If the search string **text2** is longer than the input string **text1**, the input string **text1** is returned unchanged. For example:

```
REPLACE('AA', 'AAAAA' , 'B') -> 'AA'
```

If no occurrence of the search string **text2** is found, the input string **text1** is returned unchanged. For example:

```
REPLACE( 'AAAAAA','B','C')  -> 'AAAAAA
```

If required occurrence **int1** is not found the input string **text1** is returned unchanged. For example:

```
REPLACE('AAAAAA', 'A', 'B', 10 )   -> 'AAAAAA'
```

If the number of replacements required **int2** is greater than the actual number of occurrence from the specified start occurrence, replacements are made up to the end of the string (or beginning in backwards mode). For example:

```
REPLACE('AAAAAA', 'A', 'B', 2, 8) ->
'ABBBBB'
REPLACE ('AAAAAA', 'A', 'B', -3, 8) ->
'BBBBAA'
```

STRING

Synopsis

```
STRing ( any scalar type )  -> text
STRing ( number , text1 )   -> text
STRing ( pos , text1 )      -> text
```

Description

Turns a value into a text string.

With a single argument the STRING function can be applied to the following scalar data types:

- Numeric
- Logical
- Id
- Position
- Direction
- Orientation

With only one argument, decimal places are output to give a maximum of six significant figures. Trailing zeros are always removed in this case.

With two arguments the data type may be either numeric (scalar) or position or direction. With two arguments, convert a number or position into a text string using the format described by **text1**, which may take any of the values between ’D0’ and ’D6’ (or ’d0’ and ’d6’), where the number indicates the number of decimal places.

For numbers, STRING always outputs values as millimetres. If unit conversion is needed then the DIST function should be used. For positions, the current distance units are used.

Side Effects

None.

Example

```
STRING ( 1 ) -> ’1’
STRING ( 1 , ’D3’ ) -> ’1.000’
STRING ( 1.23456789 ) -> ’1.23457’
STRING(1.1230000) ->’1.123’
STRING ( 1.23456789 , ’D3’ ) -> ’1.235’
STRING (9*9 LT 100) -> ’TRUE’
STRING (OWN OF CE) -> ’/PIPE1’
STRING(POS) -> ’W1000 N20000 U18000’
STRING(POS, ’D4’ ) -> ’W10000.1234 N20000.1234
U18000.1234’
STRING(HDIR OF /PIPE1-1) -> ’D’
STRING(E 22.0125 N, ’D2’) -> ’E 22.01 N’
STRING (ORI OF NEXT) -> ’Y IS D AND Z IS U’
```

Errors

SUBSTRING and DSUBSTRING

Synopsis

```
SUBString ( text1 ,            -> text
number1 )

SUBString ( text1 ,            -> text
number1 , number2 )

DSUBString ( text1 ,           -> text
number1 )

DSUBString ( text1 ,           -> text
number1 , number2 )
```

Description

With two arguments, return the substring of **text1** beginning at the position **number1** to the end of **text1**.

With three arguments, return the substring of **text1** beginning at the position **number1** and of length **number2**. If **number1** is negative, then counting of characters starts from the RHS of the input string. If **number2** is negative, then characters up to and including the start position are returned.

DSUBSTRING used with characters which have a displayed width that is different from standard characters, such as Japanese.

If the chosen range is outside the original string, an empty string is returned

Side Effects

None.

Example

```
SUBSTRING ( 'abcdef' , 3 ) -> 'cdef'
SUBSTRING ( 'abcdef' ,-3 ) -> 'abcd'
SUBSTRING ( 'abcdef' , 3 , 2 ) -> 'cd'
SUBSTRING ( 'abcdef' , -3, 2 ) -> 'de'
SUBSTRING ( 'abcdef' , 3 , -2 ) -> 'bc'
SUBSTRING ( 'abcdef' , 10 ) -> ''
SUBSTRING ( 'abcdef' , -10 , 2 ) -> 'ab'
```

Errors

None.

TRIM

Synopsis

```
TRIM ( text1 )                 -> text
TRIM ( text1, text2 )          -> text
TRIM ( text1, text2, text3 )   -> text
```

Description	When only one argument is supplied, TRIM removes all spaces to the left (leading) and right (trailing) of **text1** and returns the answer in text. When two arguments are supplied, **text2** specifies where the spaces should be removed from: either ’L’ or ’l’ for left, ’R’ or ’r’ for right, and ’M’ or ’m’ for multiple (where multiple occurrences of blanks are squeezed to a single spaces) or any combination of the three key letters. So the default is ’LR’ when this field is omitted. When the third argument **text3** is also supplied, this should only be a single character which overrides the space character as the character being trimmed.
Side Effects	None.

Example	`TRIM ( ’ How now, brown cow ’, ’LRM’ ) -> ’How now, brown cow’`
	
	`TRIM ( ’10.3000’, ’R’, ’0’ ) -> ’10.3’`

Errors	None.

VTEXT

VTEXT is used for the late evaluation of variables.

Synopsis	`VTEXT ( variable-name )`	`-> text`
	`VTEXT ( variable-name , number )`	`-> text`

Description	With one argument, it gets the value of the scalar variable or the value of the array variable element. With two arguments, it gets the value of the element corresponding to the index number. The value is returned as a text string. See also VLOGICAL used for late evaluation when a logical result is required, and VVALUE used for late evaluation when a numeric result is required.
Side Effects	If the scalar variable, the array variable or the array variable element does not exist, the result is undefined.

Example	`VTEXT ( !var ) -> ’hello’`
	`VTEXT ( !array[1] ) -> ’1.00’`
	`VTEXT ( !array , 2 ) -> ’0.00’`

Errors	Errors Scalar variable may not be indexed (e.g. **VTEXT (!var[1])**). Array variable must have an index (e.g. **VTEXT (!array)**).

9.7 Late Evaluation of Variables in Expressions

The functions VVALUE, VLOGICAL and VTEXT are used for late evaluation of PML variables, that is, they enable you to specify PML variables in expressions which will not be evaluated until the expression is evaluated. For example, when you are creating a report template, you are actually creating a macro which will run when a report is generated. All variables in a report template must therefore be preceded by a suitable late evaluation operator; otherwise the system will try to substitute a value for the variable when it is entered on the form. The difference between the operators is the type of output. VVALUE is used to output a numeric value, VLOGICAL to output a logical variable and VTEXT to output a text variable.

9.8 Attributes in Expressions

All attributes and pseudo-attributes may be recognised within expressions. Optionally they may be followed by 'OF' to denote a different element to the current one; e.g. POS OF / VESS1. Brackets may be used to denote an element of an array, for example `DESP[8 + 1]` for the ninth value of DESP. Since syntax clashes are possible, the keyword ATTRIB may be used to denote that an attribute follows. For example, ATTRIB E will denote the pseudo-attribute EAST as opposed to the start of a position or direction. Attributes are described in the *Data Model Reference Manual.*

9.9 Querying Expressions

All expressions may be queried. Arrays are always concatenated into a single variable. Imperial values are always output as inch to variables. This preserves maximum accuracy. To output in FINCH, then the DISTANCE function must be used. In general expression do not have to be enclosed in brackets, but to be sure that other queries are not picked up by mistake then it is advisable to do so.

Particular queries that could lead to confusion are those available both outside and inside expressions. These are:

- `Q PPOINT n`
- `Q POS or cartesian position`
- `Q ORI or cartesian orientation`

The functionality may vary between outside and inside expression queries. For example, 'Q N 100 FROM /POSS' is not valid. It must be entered as Q N 100 FROM /POSS).

9.10 Units in Expressions

When a user enters a literal value then the units are not necessarily known. The units for PML variables are also unknown. Where units are known, then all internal values are set to mm. The value is then converted to the target (local) units on assignment to a variable or on output.

To cope with 'unknown' units each value remembers its original units internally. An attempt is then made to allow for 'unknown' units across operators.

The internal settings for units are as follows:

Setting	Comments
NONE	No units. e.g. attribute OBS.
UNKN	Unknown units. e.g. 10.
MM	Dist/bore attribute if units are MM, or literal e.g. 10 mm.
INCH	Dist/bore attribute if units are INCH/ FINCH, or literal e.g. 10’.
SQIN	Multiply two INCH values together, or literal e.g. 10 sq in.
CUIN	Multiply SQIN by INCH, or literal e.g. 10 cu in.

On comparison, addition or subtraction of two values the following assumptions are made. If one of the units is unknown and the other is anything other than UNKN, then the unknown value is assumed to have the same units as the known units. A suitable conversion is then done if the known units is INCH or SQIN or CUIN.

For example:

```
(XLEN GT 10).
```

If we are working in distance units of inches, it is known that XLEN is a distance value. Internally the value is held in mm, but the units are held as INCH. The units for ’10’ are held as unknown. On doing the comparison, the ’10’ is assumed to be inches and thus multiplied by 25.4 to ensure that the comparison works as expected.

Special action is also taken to preserve the correct units across multiplication, division, POWER and SQRT, in particular the maintenance of SQIN and CUIN. In these situations, units of %UNKN are treated as none. For example`, (10 * XLEN)` is assumed to result in INCH rather than SQIN. An exception is made when a reciprocal would result from division. For example: for `(10 / XLEN)` we assume that the 10 is in inches rather than none.

9.11 Precision of Comparisons

To allow for small losses of accuracy, the following tolerances are used.

Object	Tolerance
Number	Tolerance factor of 0.000001. In other words, if the difference between two reals is not greater than 0.000001* (maximum of the two values) then the values are considered to be equal. e.g. • (1.000001 GT 1) is FALSE as it considers 1.000001; and 1 to be equal; • (1.000002 GT 1) is TRUE.
Position	Considered to be equal if within 0.5 mm of one another.
Direction or Orientation	Considered to be equal if values are within 0.005.

9.12 Undefined Values

In order to permit expressions like `((DIAM GT 200.0) OR (TYPE EQ ’BOX’))`, expressions must be able to cope with undefined values. Generally, applying an operator to one or more undefined arguments has an undefined result.

Two exceptions are: the use of the AND operator with a false argument, will result in FALSE, regardless of whether or not the remainder of the arguments are defined; and OR which returns TRUE if any of its arguments is TRUE. For example, consider applying the above expression when the current element is a box. DIAM is undefined; therefore `(DIAM GT 200.0)` is also undefined. However, `(TYPE EQ ’BOX’)` is certainly true and so the final result of the whole expression evaluates to TRUE.

An undefined result occurs when:

- One of the operands or arguments of a function (except some cases of AND and OR) is undefined.
- An attribute is unavailable for the corresponding element (e.g.`’DIAM OF OWNER’` when the current element is a box).
- An element is undefined (e.g. `’OWNER’` when the current element is the WORLD).
- An attribute is unset (e.g. text attribute or UDA of length 0).
- A variable is undefined (e.g. `’VVAL(!ARC6)’` where **!ARC6** has never been initialised).
- Two position constants are compared with GT, GE, LT or LE and they have no common coordinates (e.g. `’N10 EQ E5’`).
- If the result of the whole expression is undefined, an error occurs.

9.13 Unset Values

A particular class of undefined values are unset values. The concept exists for attributes which are valid for a given element, but for which no value has been assigned. Typically these may be elements of an array, or 'word' attributes. References of value =0/0 are also treated as unset.

Unset values are propagated as for undefined values (except for Boolean operations- see below). Undefined values take precedence over unset. There is a specific logical function UNSET to test if a values is unset.

Across comparisons, unset values are not propagated, but are treated as follows:

Operator	**When Applied to an UNSET**
EQ, GT, GE, LT, LE	Results in FALSE.
NE	Results in TRUE.
OR , AND	Values are treated as FALSE.

For example, if **DESP(2)** and **LVAL(3)** are unset then:

```
(DESP(2) GT 99) -> False
(DESP(2) NE 33) -> True
(:LVAL(3) AND TRUE) -> False
```

10 Using Rules to Define Attribute Settings

Rather than being set explicitly, the values of some types of attribute can be specified in terms of rules; that is, expressions from which the attribute values can be evaluated. Rules can be set only for attributes of the following types (including user-defined attributes): text, scalar (integer, real or logical), position, orientation, direction; they cannot be set for reference attributes. A **static** rule will change the attribute setting only when verified or executed explicitly, whereas a **dynamic** rule will update the attribute setting whenever any part of the expression changes (the default type is static).

10.1 Setting Attribute Rules

Lets you set a rule for the value of a single named attribute. The rule may contain any valid expression of the type applicable to the attribute setting.

Examples:

```
RULE SET ZLEN (XLEN + YLEN)
```

Sets rule that ZLEN of the current element is the sum of its XLEN and YLEN values. The ZLEN will be updated to reflect changes to XLEN or YLEN only when the rule is verified or executed (i.e. it is a static rule).

```
RULE SET XLEN DYNAM (YLEN + 2)
```

XLEN will be updated automatically whenever YLEN is changed.

```
RULE SET POS (N300 E400 U500) ON ALL BOX FOR /PUMP1
```

Sets rule for position attribute for all boxes in /PUMP1

```
RULE SET POS DYNAM (N100 FROM /BOX2 )
```

If BOX2 moves, the element with this attribute rule will move with it automatically. (Note space between last character of element name and closing parenthesis.)

Command Syntax:

```
>- RULE SET - attribute_name -+- STAtic --.
                              |           |
                              |- DYNamic -|
                              |           |
                                            ‘-----------+- <expre> -+- ON -.
                                                                    |       |
                                                                    ‘------+-.
                                                                             |
                                                                      .------’
                                                                      |
                                                                      ‘-+- <selatt> -.
                                                                        |             |
                                                                        ‘------------+->
```

Querying:

`Q ATT`	Displays all attribute values and all rules for the current element.
`Q RULES`	Displays all rules for current element.
`Q RUL OF XLEN`	Displays rule for XLEN attribute of current element.

10.2 Verifying Attribute Rules

When a rule is verified, the expression held in the rule is evaluated and both the result of the evaluation and the current value of the attribute are displayed.

Examples:

```
RULE VERIFY ALL
```

Verifies all rules for the current element.

```
RULE VER HEIG ON CYLI 1 FOR /PUMP1
```

Verifies rule for height attribute on first cylinder of /PUMP1.

Command Syntax:

```
>-- RULE VERify --+-- attribute_name --.
                  |                    |
                  ‘-- ALL -------------+-- ON --.
                                       |        |
                                       ‘--------+-- <selatt> --.
                                                |              |
                                                ‘--------------+-->
```

10.3 Executing Attribute Rules

When a rule is executed, the expression held in the rule is evaluated and the value of the attribute is replaced by the result of the evaluation.

Examples:

```
RULE EXECUTE :TEMP1
```

Executes rule for uda :TEMP1 for the current element.

```
RULE EXE ALL ON ALL BOX FOR /PUMP1
```

Executes all rules for all boxes owned by /PUMP1.

Command Syntax:

```
>-- RULE EXEcute --+-- attribute_name --.
                   |                    |
                   '-- ALL -------------+-- ON --.
                                        |        |
                                        '--------+-- <selatt> --.
                                                 |              |
                                                 '--------------+->
```

10.4 Deleting Attribute Rules

Lets you delete one or more rules for the current element or for specified elements.

Examples:

```
RULE DELETE ALL
```

Deletes all rules for the current element.

```
RULE DEL ALL ON ALL FOR /PUMP1
```

Deletes all rules for all primitives owned by /PUMP1.

Command Syntax:

```
>-- RULE DELete --+-- attribute_name --.
                  |                    |
                  '-- ALL -------------+-- ON --.
                                       |        |
                                       '--------+-- <selatt> --.
                                                |              |
                                                '--------------+-->
```

10.5 Rules for Arrays

Rules can be set for array attributes by using the NUM syntax, e.g. the following sets rules for the first 3 design parameters.

```
RUL SET DESP NUM 1 ( DESP(1) OF /RULE-SCTN )
RUL SET DESP NUM 2 ( DESP(4) OF /RULE-SCTN )
RUL SET DESP NUM 3 ( 100 + DESP(2) OF PREV )
```

11 Collections

You can create an array which includes a number of elements which all satisfy specific selection criteria, as defined by yourself. This is a useful way of collecting information on particular elements. You use the syntax:

```
VAR !Array  COLLECT selection criteria
```

!Array is the name of the array that will be created to contain the elements selected.

The following general criteria can be used to define the selection:

- A class of elements or element types.
- A logical expression to be satisfied at all selected elements.
- A physical volume in which all selected elements must lie.
- A point in the hierarchy below which all selected elements must lie.
  All criteria (except for class) are optional.

Class is essentially a list of element types (or possibly of actual elements). This list can be optionally qualified to indicate whether members should be included, or whether only ‘items’ (that is, the lowest level components in the hierarchy below a given element) should be included.

For example:

Command	Effect
`ALL`	Selects all elements
`ALL FRMW`	Selects all framework elements
`ALL BRANCH MEMBERS`	Selects all piping components
`ITEMS OF EQUI /VESS1`	Selects all primitives below **/VESS1**
`( /PIPE1 /PIPE2 )`	Selects only **/PIPE1** and **/PIPE2**.

The command:

```
VAR !PIPECOMPS COLLECT ALL BRANCH MEMBERS
```

Would create the array **!PIPECOMPS** and set it to contain the reference numbers of every piping component in the MDB.

Logical expressions, which return **TRUE** or **FALSE**, can be used. They are most likely to be used to check the value of an attribute for collection. The `WITH` or `WHERE` options introduce the expression. For example:

```
VAR !LENGTHS COLLECT ALL WITH ( XLEN * YLEN 8 ZLEN GT 1000 )
```

would collect all elements for which the attributes **XLEN**, **YLEN** and **ZLEN** match the criteria in the array **!LENGTHS**.

A **volume** is defined by the `WITHIN` keyword. You can define the volume either in terms of two diagonally opposite points of an enclosing box, or as a volume around an element (with an optional clearance around the box which contains the element). For example:

```
VAR !VOLUME COLLECT ALL WITHIN W800N17000U0 TO W1400N13500U1200
```

collects all elements in the defined volume into the array **!VOLUME**.

```
VAR !P COLLECT ALL PIPE EXCLUSIVE WITHIN VOLUME /PUMP1 1500
```

collects all piping components within the volume defined by a box ‘drawn’ 1500 mm around **/PUMP1** and puts them into the array **!P**. The `EXCLUSIVE` keyword indicates that only the chosen elements exclusively within the given volume are to be selected.

In Marine there are structural design data, termed DESIGN, and detailed design data, termed PRODUCTION. These two sets of data represent the same model and occupy the same 3D space. For a volumetric query you only want one of the sets of data returned.

These two options allow you to choose which set of data will be returned by the volumetric query.

Example:

`Q VOLUMEOPTION HULL DESIGN`	Returns DESIGN data
`Q VOLUMEOPTION HULL PRODUCTION`	Returns PRODUCTION data

Command Syntax

```
>--- VOLUMEOPTION ---+--- HULL DESIGN ---.
                     |                   |
                     |                   |
                     ‘- HULL PRODUCTION -+--- ON  ---.
                                         |           |
                                         |           |
                                         ‘--- OFF ---+--->
```

Hierarchy criteria can be defined by the `FOR` keyword. It identifies a list of elements below which all selected elements must occur. You can also include an exclusion list. For example:

```
VAR !BRANCH COLLECT ALL BRANCH MEMBERS FOR /PIPE1 /PIPE2
EXCLUDE BRAN 1 OF /PIPE2
```

You can **append** the results of such a collection to an existing array using the `APPEND` keyword. For example:

```
VAR !BENDS APPEND COLLECT ALL ELBOWS
```

Would add the references for all elbows to the array **!BENDS**.

You can also **overwrite** elements in the array by specifying the first index in the array which you want to be overwritten. The specified index, and the indexes following it, will be overwritten by the results. For example:

```
VAR !BENDS[99] COLLECT ALL ELBOWS
```

Would place the reference for the first **ELBOW** selected at position 99 in the array **!BENDS**, overwriting any existing data, and subsequent selections in the array elements that follow.

If you specify more than one criteria, the specifications must be in the above order Some more examples:

`ALL`	Selects all elements
`ALL FRMW`	Selects all framework elements
`ALL BRANCH MEMBERS`	Selects all piping components
`ITEMS OF EQUI /VESS1`	Selects all primitives below /VESS1
`( /PIPE1 /PIPE2 )`	Selects just /PIPE1 and /PIPE2
`ALL WITH (XLEN GT 1000 )`	Selects all elements where XLEN is greater than 1000mm
`ALL WITHIN W8000N17000U1000 TO W1400N13500U1200`	Selects all elements within the defined volume
`ALL PIPE WITHIN VOLUME /PIPE1 1500`	Selects all piping elements within a volume defined as a box drawn around /PIPE1, with a clearance of 1500mm between the edges of /PIPE1 and the volume box.

Note: This selection mechanism is a very powerful tool for searching whole databases and MDBs. However, if you're not careful the selection process could be very time consuming and tie up a lot of computer resource. Therefore, it is important that selection is performed as efficiently as possible. Marine tries to apply the above criteria so that the fastest condition is applied first and the most expensive is left to last.

Typically, the expression is the slowest condition to evaluate, so it is important to limit the selection as much as possible. For instance, take the example which appeared above:

```
ENHANCE ALL WITH ( XLEN * YLEN * ZLEN GT 100 0 )
```

Since only BOXes (and NBOXes) meet this criterion it would be sensible to limit the search by specifying an appropriate class:

```
ENHANCE ALL BOX WITH ( XLEN * YLEN * ZLEN GT 100 0 )
```

This cuts the time to execute the selection. This is because the selection system knows that BOXes only occur in DESI databases. Therefore it does not search other types of database. It also knows where boxes are in the hierarchy, and so does not search unnecessary elements.

Even greater performance savings can be gained by explicitly limiting the elements which have to be visited by the search:

```
ENHANCE ALL BOX WITH ( XLEN * YLEN * ZLEN GT 1000 ) FOR /*
```

By default, the entire MDB is searched. But by specifying a hierarchy criterion, the selection time can be cut considerably.

Limiting the volume of the search also cuts the number of elements which have to be checked. However, it should be noted that this criterion is applied by determining whether element limit boxes fall within the specified volume, using the spatial map. This is a fast approach, but is not meant to provide the same accuracy as is used in on line clashing.

```
VAR IPIPECOMPS COLLECT ALL BRANCH MEMBERS
```

will set up the array IPIPECOMPS to contain the reference numbers of every piping component in the MDB, e.g.

```
IPIPECOMPS [1]   = '=20/302'
IPIPECOMPS [2]   = '=20/303'
IPIPECOMPS [3]   = '=20/304'
           .
           .
           .
IPIPECOMPS [354] = '=25/510'
```

Every flange could then be extracted as follows:

```
VAR !FLANGES COLLECT (ALL FLANGES) FROM IPIPECOMPS
```

and then enhanced (highlighted):

```
ENHANCE ALL FROM !FLANGES
```

This could alternatively be performed in one step:

```
ENHANCE ALL FLANGES FROM IPIPECOMPS
```

Collections may be joined or concatenated by preceding the COLLECT keyword by APPEND:

```
VAR !BENDS APPEND COLLECT ALL ELBOWS
```

Alternatively, the following:

```
VAR !LIST[99] COLLECT ALL SLCY
```

would place the reference of the first SLCY at position 99 in !LIST overwriting any data that already exists at that and subsequent elements of the array.

If a selection contains elements of type TUBIng, then the collection describes it as the Leave Tube of an existing database element:

```
VAR !TUBING COLLECT (ALL TUBI) FOR /*
```

Then !TUBING would contain something like the following:

```
!TUBING[1]      'IL TUB OF =20/302'

!TUBING[2]      'IL TUB OF =20/303'
```

The evaluate command allows an expression to be evaluation for all members of a collection.

The syntax is:

VAR !variable EVALUATE expression For selection e.g. to get the description of all equipment you can do:

```
var !cln collect all EQUI
var !name evaluate (description) for all from !cln
```

or

```
var !name evaluate (description) for all EQUI
```

The PML collection object can also be used as an alternative to the COLLECTION syntax. This object is described in the *Software Customisation Reference Manual*.

12 Comparisons Across Sessions and Stamps

12.1 Change Management

You can query the following aspects of the history of modifications to the current database:

- When and by whom an element or attribute was last modified.
- A complete history of the sessions in which an element or attribute has been modified.
- Details of a given session.
- The session number for a given date.

12.1.1 Querying the Last Modification to an Element or Attribute

Lets you query details of the most recent change to a given element or attribute.

Examples:

`Q LASTMOD`	Gives date for last modification to current element.
`Q SESSMOD`	Gives session number for last modification to current element.
`Q USERMOD`	Gives name of user who last modified current element.
`Q LASTMOD HIER`	Gives dates for last modifications to current element and its members.
`Q LASTMOD XLEN`	Gives date for last modification to XLEN attribute of current element.

Command Syntax:

```
Q --+-- LASTMod --.
    |             |
    |-- SESSMod --|
    |             |
    ‘-- USERMod --+--+-- <selatt> --.
                  |  |             |
                  |  ‘-------------+-- HIERarchy --.
                  |                |               |
                  |                ‘---------------+-->
                  |
                  ‘-- attribute_name -->
```

12.1.2 Querying the Session History for an Element or Attribute

Lets you query modification history for a given attribute; i.e. session numbers during which the attribute was modified.

Examples:

`Q HISTORY DIAM`	Gives all sessions in which DIAM attribute was modified.

Note: HISTORY is an array type pseudo-attribute, so that qualifying positions may be appended to query specific occurrences in the modification history. For example:

```
Q HISTORY[2] DIAM
```

gives second most recent session in which DIAM attribute was modified.

Note: History records are restricted to a maximum of 120 sessions.

Command Syntax:

```
Q HISTORY attribute_name
```

12.1.3 Querying Details of a Specific Session

Lets you query details of any specific session. This is particularly useful to get details of sessions listed by a HISTORY command.

Examples:

`Q SESSCOMM 58`	Gives comment text associated with session 58
`Q SESSUSER 58`	Gives name of user responsible for session 58.
`Q SESSDATE 58`	Gives date and time at which session 58 was created.

Note: All session queries are for the current DB.

Command Syntax:

```
Q --+-- SESSComment --.
    |                 |
    |-- SESSUser -----|
    |                 |
    ‘-- SESSDate -----+-- integer -->
```

12.1.4 Querying Session Number for a Given Time

Lets you query which session was current at a given time. (This is the inverse of the Q SESSDATE option described in *Querying Details of a Specific Session*.)

Examples:

```
Q SESSION ON 12:00 22 August 1995 Q SESSION ON 9 /9 /96
```

Time defaults to 23:59, so returns last session number on given date.

Command Syntax:

```
Q SESSION ON <date>
```

where <date> is a standard syntax graph. Remember that <date> actually specifies a time (to the nearest minute), so take care if you use any defaults here.

12.2 Comparison Date

It is only by comparing a drawing at two states or sessions that it is possible to determine what has changed. Using the current state of the drawing as one state we must then reference an earlier state in order to make the comparison. We do this by specifying a Comparison Date (COMPDATE), that is, the drawing state at a time that we wish to use as a baseline or datum.

12.2.1 Setting the Comparison Date

You can enter a comparison date, either for the entire MDB or an individual DB. For individual DBs, you can also enter a specific session number and extract number.

Examples:

```
SETCOMPDATE 31 March 2002

SETCOMPDATE STAMP /STAMP1

SETCOMPDATE NOW (will compare against the start values)

SETCOMPDATE FOR CTBATEST/DESI to session 99

SETCOMPDATE FOR CTBATEST/DESI to EXTRACT (this will compare
against the parent)

SETCOMPDATE FOR CTBATEST/DESI to CTBATEST/MASTER (CTBATEST/
MASTER must be up the extract hierarchy)
```

Command Syntax:

```
-SETCOMPDATE--|---date->
              | --STAMP------name->
              ‘-FOR--DB--dbname--TO--|--date-->
                                     |--Session -int-|--|-EXTRACT--|—- int---->
                                     ‘---------------‘  ‘-->       |-- Dbname->
                                                                   ‘---------->
```

The ‘date’ subgraph takes the keyword NOW This in effect sets the comparison date to the start of the session. This can be useful for querying the original value of an attribute.

The EXTRACT keyword sets the comparison to an extract DB. This extract DB must be one further up the extract hierarchy. If the EXTRACT keyword is used by itself, the comparison is set to the parent extract. Thus this enables you to find out what has been changed in this extract.

12.2.2 Querying the Comparison Date

The query will return the comparison session number or extract number for a DB.

Examples:

`Q COMPDATE  EXTRACT FOR DB CTBATEST/DESI`	to get extract
`Q COMPDATE COMPDATE SESSION FOR DB CTBATEST/DESI`	to get session
`Q COMPDATE DATE`	to get date
`Q COMPDATE STAMP`	to get stamp

Note: Note that if a stamp is used to set the comparison date, this will set the comparison session for each database within the stamp. It will also reset any comparison dates set previously.

The query for the comparison date will only return a value if the COMPDATE was set using a single date. Otherwise it will return ‘unset’. Similarly querying a stamp is only valid if the COMPDATE was set using a stamp.

Command Syntax:

```
Q ----------|-COMPDATE-|--SESSION--|--FOR---dbname--->
VAR -vname--‘          |—EXTRACT---´
                       |----DATE--------->
                       ‘----STAMP-------->
```

12.2.3 MODIFIED Function

For the more sophisticated queries relating to modifications, the MODIFIED function tells you if the given element has changed since the comparison date. This function is not implemented within PML2 expressions.

Examples:

To return true if element has changed at all since the comparison date use:

```
Q MODIFIED()
```

It will also return true if the element has been created since the comparison date.

To return true if POS or ORI have been modified since the comparison date use:

```
Q MODIFIED(POS,ORI)
```

Examples:

To return true if the position of P1 has changed use.

```
Q MODIFIED(P1 POS)
```

You may follow each attribute name with the qualifying keywords below.

To check this element and members use:

```
OFFSPRING
```

To check all elements for which this element represents the significant one use:

```
SIGNIF
```

To check all elements for which this element represents the primary one use:

```
PRIMARY
```

To check this element and everything below (descendants):

```
DESCENDANTS
```

You can use the keywords below on their own to test for any attribute change. e.g. to return true if any geometry for item or any descendants have changed use:

```
Q MODIFIED(GEOM DESCENDANTS)
```

To return true if any element for which this element is primary, has changed use:

```
Q MODIFIED(PRIMARY)
```

You may use the 'OF' syntax as for attributes. e.g. to return true if /PIPE1 has been modified since the comparison date use:

```
Q MODIFIED() OF /PIPE1
```

You may put the new functions anywhere within an Outfitting PML1 expression. i.e. after Q/Var and within collections. e.g.

```
Q (BUIL OR MODIFIED() OR ELECREC OF NEXT )
```

Command Syntax:

```
                                   .-------------------------------------.
                                  /                                       |
>-MODIFIED-(-+--attname-------|--*--DESCENDANTS--+--+-comma--+--attname--´
             |                   |               |  |
             |--DESCENDANTS--.   |-- SIGNIFICANT-|  |
             |               |   |               |  |
             |--SIGNIFICANT--|   |--PRIMARY----- |  |
             |               |   |               |  |
             |--PRIMARY------|   |--MEMBERS------|  |
             |               |   |               |  |
             |--MEMBERS------|   ´---------------´  |
             |               |                      |
             |               |                      |
             |               |                      |
             ´---------------+----------------------+--+--) ---OF --id-->
                                                       |
                                                       ´-->
```

12.2.4 CREATED Function

Determine if an element has changed since the Comparison Date. The functionality of CREATED() is identical to using the pseudo attribute ELECREC.

Examples:

```
Q ( CREATED() )
```

12.2.5 DELETED Function

Determine if an element has been deleted since the Comparison Date. The functionality of DELETED() is identical to using the pseudo attribute ELEDELC.

Examples:

`Q ( DELETED() )-` returns deleted since comparison date

However if the element has been deleted then you cannot have navigated to it in the first place, hence DELETED() by itself will always be true. There are two ways around this.

Either include the element's reference number e.g.:

```
    Q (DELETED() of =15752/234 )
```

Or use it as part of the 'old' syntax. e.g.:

```
    Q OLD (DELETED() of /VESS2)
```

12.2.6 GEOM, CATTEXT, and CATMOD Special Attributes

There are three new special attributes 'GEOM', 'CATTEXT', and CATMOD (previously called 'CATA').

GEOM Special Attribute

The GEOM attribute returns true if any aspect of the evaluated geometry has changed.

The definition of evaluated geometry change includes:

- Any dimension of a primitive has changed
- Any ppoint changes
- Pos/ori change

  The level information used to determine the geometry is set by the 'REPRE MASS' command. The 'REPRE MASS' command is also available in ISODRAFT.

CATTEXT Special Attribute

This will return true if any part of the evaluated detail or material text has changed.

CATMOD Special Attribute

Special attribute CATMOD will return true if any value in the catalogue has changed. i.e.

- SPREF
- Changes to SPCO element

- Changes to COMP element
- Changes to any PTSE, GMSE, ppoint, geometry elements
- Changes to any dataset elements
- Changes in DTEXT,MTEXT elements

  Note that there is a subtle difference between CATMOD and the other two: the CATTEXT and GEOM keywords work on the evaluated values.

Thus it is possible that the geometry element has changed but the GEOM keyword returns false, e.g. a UDA value may have changed, but this has no effect on the evaluated geometry.

The CATMOD keyword on the other hand will return true for any change.

You can use the CATMOD keyword on any element. It will return ‘false’ if the element does not have a SPREF or CATREF reference pointing into the catalogue database. It will return ‘true’ if the element has a SPREF or CATREF attribute and either (a) this reference attribute has itself changed in value or (b) the catalogue element pointed at, or any catalogue element owned by or pointed at by this element, either directly or indirectly, has changed in any way.

The exception is that elements pointed at by UDA’s are not compared, although the value of the UDA itself is checked. Thus if a reference valued UDA has been changed then this will count as a change, but if only the element *pointed at* has changed, then this will not count.

12.2.7 Querying Any Attribute at the Comparison Date

The ‘OLD’ syntax enables you to query any attribute at the comparison date.

You can use the syntax in front of any expression or attribute. The whole expression will then be evaluated at the comparison date. e.g.

```
Q OLD XLEN
```

If a name is given, the name will be for the item at the comparison date, not now. Thus values of deleted items may be accessed. e.g.

```
Q OLD REF OF /OLDPIPE
```

Where /OLDPIPE no longer exists.

The ‘OLD’ syntax may also be used after ‘VAR’. This includes collections e.g.

```
VAR !PIPES OLD COLLECT ALL PIPES
```

This would return a collection of all PIPES at the old version.

If the functions MODIFIED, CREATED, DELETED are used on the old version then the comparison is made with the current version.

For example to get a list of deleted pipes between the comparison date and now, then the following collection could be used. e.g.:

```
VAR !PIPES OLD COLLECT ALL PIPES WITH ( DELETED() )
```

There is also a pseudo attribute, DSESS. that returns the session number when an element was deleted. i.e. having got the deleted PIPES from the previous query, we can now find out when they were deleted.

12.3 Comparing Database Changes

12.3.1 Comparing Database States at Different Times

You can compare details of your current database settings with the corresponding settings at a specified earlier time and generate a report listing all differences. The types of change reported include:

- Creation and deletion of elements.
- Changes to the attribute settings of elements.
- Changes in the list order for BRANCH, POGON, DRAWI and BOUND elements.

Keywords:

DIFFERENCE SINCE

Description:

Lets you report all changes to one or more specified database elements since an earlier version of that database. The output is in the form of a report listing all elements and attributes which have changed, with their old and new values. The report can be sent to a file by using the ALPHA FILE or ALPHA LOG commands.

Note: The database states are compared between SAVEWORK operations. For example, if you last saved your design changes at 9:30 and ask for a comparison since 10:00, the current settings will be compared with those at 9:30.

Examples:

`DIFFERENCE ALL BRANCH FOR /ATEST SINCE 21 JANUARY`	
`DIFF CE SINCE 10:00`	Assumes current day.
`DIFF /ZONE`	Compares current settings with those at your last SAVEWORK command.
`DIFF SITE SINCE SESSION 66`	Compares current settings with those at the end of session 66 of the current database.

Command Syntax:

```
>- DIFFerence <selele> SINCE -+- <date/time> -+-----------------------.
                              |               |                       |
                              |- LATEST ------|                       |
                              |               |                       |
                              |--SESSION nn --|                       |
                              |               |                       |
                              '---------------+- EXTRACT -+- extname -|
                                                          |           |
                                                          '- extno ---+->
```

13 Output Syntax

The OUTPUT command is used to scan specified parts of the Project DBs and to output, in the form of a structured list, the data held there. The output is presented in such a way that it is both easy to interpret and suitable for reinput as design data to appropriate Outfitting modules.

Output takes the form of macro files whose contents precisely recreate the hierarchical structure of the elements currently listed in the selected DBs, including the settings of all of their attributes. Facilities are provided for controlling the precise layout of the output files and the amount of information presented. Element cross-referencing (indexing) is also available, to assist in interpreting the data. The macro is sent to a file by using the standard ALPHA FILE or ALPHA LOG commands.

You may view the output lists directly on your screen, or you may send them to text files for subsequent inspection or printing. The latter files can be read back as input to say a different constructor module form that from which the data was derived, either in the same or in a different project. You can include only the elements which have been changed since a specified time (i.e. those elements which would be listed by the DIFFERENCE command).

The macro files created can be used for the following purposes:

- To copy part of a design.
- To modify part of a design. The output macro file containing the relevant design data can be edited using operating system facilities and can then be reinput to the appropriate DB. You could use this method, for example, to change Nozzle Catalogue References in a pipework subsection.
- To transfer part of a design from one DB to another or from one project to another.
- To archive all or part of the Project DB. Listings may be read in at any future revision of Outfitting, making such as archive more secure in some respects than the Project DB itself.
- To give you a quick-reference listing of the DB contents to provide a rapid answer to a specific question (such as where a particular element is stored in the hierarchy).

The output is generated in three stages:

1. Any elements which were originally locked are unlocked. Element deletions, name changes and type changes are output. Note that reordering or insertion of elements in their owner’s members list is treated as deletion followed by creation, so that Refno attribute settings may be changed.
2. Newly created elements and all standard attribute settings are output.
3. Reference attribute settings and rules are output. Elements which were originally locked are relocked and GADD commands are included if any elements were included in Groups.

13.1 General Features of Output Lists

Output by default is in a format suitable for direct reinput to most Outfitting implementations. You may, however, modify several aspects of the output format if necessary for specific purposes.

To ensure the successful reinput of data, macro files output have the following features:

- Attributes and other data which are output for information only, and which cannot be reinput (such as the time and date of the report, the element owners, etc.) are enclosed by the delimiter codes $ (and $). They are, in consequence, treated as comments on reinput and are ignored as data.
- Cross-reference attributes (for example HREF, CREF) are enclosed between the comment delimiters $ (and $) where they occur with their elements, and are output again collectively at the end of the list. This ensures that all elements which are referenced in the list are present in the database before any references to them are set.
- An END command is output after each new element has been created. This moves the Current Element pointer up to its original level in the hierarchy, ensuring that those elements (such as BOX) which may appear at two different levels in the hierarchy are reinput at the correct level.
- The single apostrophe character ‘ is output as ‘ ‘ when it occurs within text.
- The units of measurement used to output dimensional and positional attributes are, where possible, appropriate to the input requirements of the modules in which they will be used. For example, the Bolting attributes BDIA and LENG are always in millimetres, so as to be consistent with the input to PARAGON.
- Millimetre coordinates and dimensions are normally listed to three decimal plates (i.e. to an accuracy of 0.001mm), and so a previous list may be used to check design data which has been created with a lower precision in another module. This could be useful, for example, in assessing a misalignment reported by a data consistency check.

Note: Numerical accuracy deteriorates after six significant figures. The number of decimal places output can be varied using the PRECISION command.

- ORI and ANGLE attributes are listed to four decimal plates. Orientations are output as angles using the ORIANGLE attribute to give more accurate output, and (as comment text) in XYZENU axis form. For example

```
AT W17246.099 N12125.000 U4130.000 ORIA 180.000 -75.000
90.000 $ ( ORI Y IS E AND Z IS N 15.000 D $)
```

- Nozzles are treated somewhat differently to other elements in that their positions and orientations are output twice; first in Owner coordinates (for normal reinput), and secondly in Zone coordinates. The latter data enables them to be checked directly against Branch head and Tail positions, which are always stored in Zone coordinates. (Examples are given later in this section).

Note: The lengths of output lists are normally minimised by omitting all attributes which have the Outfitting default values, since the use of such values in input data generally has no effect. The omission of data in this way can, however, cause problems under some circumstances, and so may be overridden if required.

13.2 Principles and Limitations

Output listing can be used as a convenient medium for transferring data from one project to another or from one computer to another. Such lists may also be useful when a Project DB is upgraded to a newly released version of Marine. This chapter summaries the procedures which you might follow when carrying out such tasks.

The design to be transferred can include some of all of the following major categories of data:

- The Project Configuration
- Catalogue items, including Bolt Tables, Connection Compatibility Tables, etc.
- Material Properties
- Component Specifications
- Three-dimensional Design layouts
- User-Defined Attributes (UDAs)
- Drawing Libraries and Drawings

To transfer a complete Project you would normally use the ADMIN Module. You would usually use output lists only for transferring specific parts of the Catalogue, Design, Drawing (PADD) or Dictionary (UDA) data.

Output cannot be used to transfer the Project Configuration; you must use ADMIN to output the configuration in a suitable format for transfer. This operation is included in this chapter simply to show where it fits into the overall operation.

Before any transfer takes place, you should consolidate the Project DB by removing all DB copies, leaving only the Master versions. You should also ensure that individual DBs have consistent and unique contents, thus avoiding, for example, an attempt to transfer two similar (but not identical) copies of the Catalogue.

When an Outfit listing is to be input to a Project DB, it is essential that all elements which are referred to in the list, but which are specified outside the list, have already been loaded into that DB. After the transfer of data to a new Project, the element reference numbers will almost certainly be different from the original and no reliance should be placed on their meaning. As a general rule, therefore, all items which are reference (such as Catalogue Components, Specification Components, etc.) should be named.

Note particularly that, if you use Outfit to transfer data from Design, Catalogue or Drawing (PADD) DBs which include User-defined Attributes (UDAs), the reloading process will fail if there are any UDSs for which definitions are not available in the target Project of if the definition in the target project is inconsistent with the definition in the source project. For this reason, UDA definitions held in the Dictionary DBs should be transferred first.

Note: Cross-references between Branches and attached Nozzles may be lost during the transfer process. In this case, they must be reset in the newly created DB.

Note: Datal listings containing Hull elements should not be manually edited. Hull elements contain binary data and manual editing may break the consistency of the hull data model.

13.3 OUTPUT Command

The following options are available:

OUTPUT CHANGES

Incorporates INCLUDE command for named items. The item must be named in both the current and referenced session. For unnamed items, a DELETE, CREATE sequence is followed.

OUTPUT REVERSE CHANGES

This is the opposite of the current OUTPUT CHANGES command: that is the output is generated can be read back in to restore the given data to how it was at the given session.

OUTPUT CHANGES SINCE

Lets you output all changes to one or more specified database elements since an earlier version of that database. The output is in the form of a macro which can recreate the changes when run on, say, a copy of the original DB. The macro is sent to a file by using the standard ALPHA FILE or ALPHA LOG commands.

Examples:

`OUTPUT /ZONE-A`

Outputs all elements, whether or not they have ever been changed.

`OUTPUT ALL PIPE FOR /ZONE CHANGES SINCE 21 JANUARY`

Outputs all changes to named element and its members since the given date.

`OUTPUT /PIPE-100 CHANGES`

Outputs all changes to named element and its members since last SAVEWORK command.

`OUTPUT /PIPE-1 CHANGES SINCE EXTRACT`

In an extract database, outputs all changes since the extract was created.

`OUTPUT /PIPE-1 CHANGES SINCE LATEST EXTRACT`

In an extract database, outputs all changes compared with the latest version of the parent extract.

```
OUTPUT /PIPE-1 CHANGES SINCE EXTRACT 44
OUTPUT /PIPE-1 CHANGES SINCE EXTRACT PIPE/PIPE-X1
```

In an extract database, outputs all changes compared with the latest version of the given extract, which must be higher in the extract hierarchy.

```
OUTPUT /PIPE-1 CHANGES SINCE SESSION 77 EXTRACT 44
OUTPUT /PIPE-1 CHANGES SINCE OCT 2000 EXTRACT PIPE/PIPE-X1
```

In an extract database, outputs all changes compared with the given extract, which must be higher in the extract hierarchy, at the given session or date.

The macro is sent to a file by using the standard ALPHA FILE or ALPHA LOG commands.

You can also give an Outfitting session number. The database states are compared between SAVEWORK operations. For example, if you last saved your design changes at 9:30 and ask for a macro containing changes since 10:00, the macro will contain all changes since 9:30.

Command Syntax:

```
>- OUTPUT <selele> SINCE -+- <date/time> -+-----------------------.
                          |               |                       |
                          |- LATEST ------|                       |
                          |               |                       |
                          |--SESSION nn --|                       |
                          |               |                       |
                          '---------------+- EXTRACT -+- extname -|
                                                      |           |
                                                      '- extno ---+->
```

- The following options are not available with the OUTPUT CHANGES functionality. Once one (or more) of these options have been specified then REVERSE, CHANGES, etc. cannot be specified following the element to be output.

COMMENT

This specifies that certain comments will be added to the OUTPUT output. The reference number of each element will be shown immediately after the NEW (or OLD) command, the owner of each element will be given, and each reference valued attribute will be output as a comment during the first pass (normally, reference valued attributes are ignored entirely during the first pass). In addition, the position and orientation of Nozzles will be written in the coordinate system of their ZONE. All these comments will be enclosed between the standard comment delimiters, that is:

$(' comment_text '$)

TABULATE n

This specifies that the output will be tabbed by n*d spaces at the beginning of each line, where d is the depth of the element. Note that where a logical line is split over more than one physical line in the file (which can happen very easily when a short line length has been specified on the 'ALPHA FILE' command) then subsequent physical lines are not tabbed.

n must be between 0 and 6.

TABULATE 0 is equivalent to no tabbing.

INDEX

This specifies that

- Each line of the output will be numbered and
- Indexes by reference number and name will be written at the end.

As with tabulate, only logical lines will be numbered; continuation lines will not be numbered.

BRIEF

This specifies that only the NEW or OLD command line will be written, that is, no attributes will be written.

NOUDA

This specifies that user defined attributes will not be written.

ONLY NOUN or

ONLY (NOUN …. NOUN)

These options specify that only elements of the given types will be output. If more than one type is to be specified, then the list must be enclosed in brackets. At most 10 types may be specified.

PASS n

This specifies that only the first pass (element definitions; no reference valued attributes) or the second pass (reference valued definitions, connections) will be written. n must be 1 or 2.

OLDFORMAT

This specifies that element definitions will be written using the OLD (rather than the NEW) command, that is, elements are to be updated when the file is re-read.

Locate

If the LOCATE option is used and 'output /VESS1', then the output will contain the following:

```
NEW LOCATE SITE /ATEST
NEW LOCATE ZONE /ZONE.EQUIP
NEW EQUIP /VESS1
Etc.
```

REPLACE

If the replace option is used the output will contain the following:

```
NEW REPLACE EQUI /VESS1
Etc.
```

N.B. the REPLACE command will only be output for the top level element.

If both LOCATE and REPLACE options are used, the output would be:

```
NEW LOCATE SITE /ATEST
NEW LOCATE ZONE /ZONE.EQUIP
NEW REPLACE EQUIP /VESS1
```

SAMER/EF

May be specified after the element to be output, in conjunction with the various formatting options and with 'CHANGES SINCE date' and 'REVERSE'. If specified, each 'NEW' command to originally define each element will be output in the form:

1. NEW <eltype> <element> REF =rrrrr/rrrrr

   Thus when the file is read back in, the element will be assigned the same reference number as previously. Of course, this requires that the reference number is suitable in the environment in which the file will be read.

Examples:

- OUTPUT CE SAMEREF
- OUTPUT INDEX /HTEST SAMEREF
- OUTPUT /STAN.CATA CHANGES SAMEREF
- OUTPUT CE REVERSE CHANGES SAMEREF

2. The NEW command has been extended to allow a new keyword 'REF' followed by a reference number to be specified after the element name. If so specified, and the reference number is valid, then the element will be created with the reference number given.

If an invalid reference number is given, the command will be rejected. Two new error messages may occur:

- 859: Invalid reference number: =rrrrr/rrrrr
- 860: Reference number =rrrrr/rrrrr already exists

Examples:

- NEW BOX /BOX1 REF =15772/17461

13.4 Some Examples of Output

The examples of output lists in this chapter illustrate the effects of the various formatting options. Often output files are large, for illustration these have been truncated.

Each example shows the commands used to generate the style of listing following them.

Note: The examples are independent; each starts with all formatting options in their default states.

13.4.1 Full Output

The following example shows a segment of a full output. Output contains all elements and non-default attribute settings

Command

```
OUTPUT /HTEST
```

Output

```
INPUT BEGIN
NEW SITE /HTEST
POS E 0 N 0 U 1000

NEW ZONE /EQUIP
FUNC 'Equipment - above grade'
PURP EQUI

NEW EQUIPMENT /E1301
POS E 2850 N 5660 U 1470
UCOFG E 0 N 0 U 0
BUIL true
DSCO unset
PTSP unset
INSC unset

NEW CYLINDER
POS E 0 N 3299 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 960
HEIG 6598

END
NEW CYLINDER
POS E 0 N 6848 U 0
ORI Y is D and Z is N
```

```
LEVE 0 4
DIAM 1020
HEIG 500
```

13.4.2 Comment Option

Will output in the same format as Full Output but will include comments to identify sections of the output.

Command

```
OUTPUT COMMENT/HTEST
```

Output

```
INPUT BEGIN
NEW SITE /HTEST $(=15772/17197$)
$( OWNE /* $)
POS E 0 N 0 U 1000

NEW ZONE /EQUIP $(=15772/17198$)
$( OWNE /HTEST $)
FUNC 'Equipment - above grade'
PURP EQUI

NEW EQUIPMENT /E1301 $(=15772/17213$)
$( OWNE /EQUIP $)
POS E 2850 N 5660 U 1470
UCOFG E 0 N 0 U 0
BUIL true
DSCO unset
PTSP unset
INSC unset

NEW CYLINDER $(=15772/17214$)
$( OWNE /E1301 $)
POS E 0 N 3299 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 960
HEIG 6598

END
NEW CYLINDER $(=15772/17215$)
$( OWNE /E1301 $)
```

```
POS E 0 N 6848 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 1020
HEIG 500
```

13.4.3 Tabulate Option

Will output with the code tab indenting

Command

```
OUTPUT TABULATE 2 /PIPES
```

Output

```
INPUT BEGIN
NEW ZONE /PIPES
  FUNC 'Piping - above grade'
  PURP PIPE

  NEW PIPE /100-B-1
    BUIL false
    SHOP false
    TEMP -100000
    PRES 0
    PSPE /A3B
    CCEN 0
    CCLA 0
    LNTP unset
    REV 0
    DUTY unset
    DSCO unset
    PTSP unset
    INSC unset
    UCOFG E 0 N 0 U 0
    DELDSG unset

    NEW BRANCH /100-B-1-B1
      BUIL false
      SHOP false
      HPOS E 12490 N 12280 U 1150
      TPOS E 4979 N 9887 U 13655
      HDIR U
      TDIR N
      UCOFG E 0 N 0 U 0
```

```
LHEA true
LTAI true
HBOR 50
TBOR 100
HCON FBD
TCON FBD
DETA true
LNTP unset
TEMP -100000
PRES 0
HSTU /A3B/PA50
CCEN 0
CCLA 0
PSPE /A3B
DUTY unset
DSCO unset
PTSP unset
INSC unset
PTNB 0
DELDSG unset
```

13.4.4 Index Option

Will include line numbers within comments at the start of each line. At the end of a file an index of refno/name is also included in comments

Output Syntax

```
OUTPUT INDEX /EQUIP
```

Output

```
 $(       1. $) INPUT BEGIN
$(       2. $) NEW ZONE /EQUIP
$(       3. $) FUNC 'Equipment - above grade'
$(       4. $) PURP EQUI

$(       5. $) NEW EQUIPMENT /E1301
$(       6. $) POS E 2850 N 5660 U 1470
$(       7. $) UCOFG E 0 N 0 U 0
$(       8. $) BUIL true
$(       9. $) DSCO unset
$(      10. $) PTSP unset
$(      11. $) INSC unset

$(      12. $) NEW CYLINDER
$(      13. $) POS E 0 N 3299 U 0
$(       14. $) ORI Y is D and Z is N
$(       15. $) LEVE 0 4
$(       16. $) DIAM 960
$(       17. $) HEIG 6598

$(       18. $) END
$(       19. $) NEW CYLINDER
$(       20. $) POS E 0 N 6848 U 0
$(       21. $) ORI Y is D and Z is N
$(       22. $) LEVE 0 4
$(       23. $) DIAM 1020
$(       24. $) HEIG 500
INPUT FINISH
$(
         --- REF  INDEX ---
   =15772/17198        ...      2
   =15772/17213        ...      5
   =15772/17214        ...     12
   =15772/17215        ...     19
   =15772/17216        ...     26
   =15772/17217        ...     33
   =15772/17218        ...     40
   =15772/17219        ...     48
   =15772/17220        ...     56
   =15772/17221        ...     64
   =15772/17222        ...     72
   =15772/17223        ...     80
   =15772/17224        ...     88
   =15772/17225        ...     96
   =15772/17226        ...    104
   =15772/17227        ...    114
   =15772/17228        ...    124
   =15772/17229        ...    134
```

13.4.5 Brief Option

Will only output element names

Output Syntax

```
OUTPUT BRIEF /STEEL
```

Output

```
INPUT BEGIN
NEW ZONE /STEEL
NEW STRUCTURE /EQUIPRACK
NEW FRMWORK /EQUIPRACK/MAIN
NEW SBFRAMEWORK /EQUIPRACK/MAIN/NODES
NEW PNODE /PNOD-009
NEW PJOINT
END
END
NEW PNODE /PNOD-010
NEW PJOINT
END
END
NEW PNODE /PNOD-012
NEW PJOINT
END
END
NEW PNODE /PNOD-013
NEW PJOINT
END
END
NEW PNODE /PNOD-014
NEW PJOINT
END
END
NEW PNODE /PNOD-015
NEW PJOINT
END
END
NEW PNODE /PNOD-016
NEW PJOINT
END
END
```

13.4.6 NOUDA Option

Will take the format of a full output, but UDA's will be suppressed.

Output syntax

```
OUTPUT NOUDA /HEATING-VENTS
```

See full output for formatting.

13.4.7 OLDFORMAT Option

Will output old commands only.

Output Syntax

```
OUTPUT OLDFORMAT /CIVIL
```

Output

```
INPUT BEGIN
OLD ZONE /CIVIL
FUNC 'Civils'
PURP CIV

OLD STRUCTURE /F1.PLANT
UCOFG E 0 N 0 U 0
BUIL false

OLD SUBSTRUCTURE /F1.PLANT.FLR
UCOFG E 0 N 0 U 0
POS E 0 S 2000 U 0
BUIL true
SHOP false

OLD PYRAMID /F1.PLANT.FLR-1
POS E 2225 N 18525 D 232.5
ORI Y is N and Z is E
LEVE 0 8
XBOT 450
YBOT 9100
XTOP 420
HEIG 3050
XOFF 15
YOFF 2150

END
```

13.4.8 ONLY option

Specify only certain elements for output. Below are examples of variations of this command

Output Syntax for Sites Only

```
OUTPUT ONLY SITE /HTEST
```

Output

```
INPUT BEGIN
NEW SITE /HTEST
POS E 0 N 0 U 1000
```

```
END
INPUT END  /HTEST
INPUT FINISH
```

Output Syntax for Branches Only

```
OUTPUT ONLY BRAN /HTEST
```

Output

```
INPUT BEGIN
NEW BRANCH /100-B-1-B1
BUIL false
SHOP false
HPOS E 12490 N 12280 U 1150
TPOS E 4979 N 9887 U 13655
HDIR U
TDIR N
UCOFG E 0 N 0 U 0
LHEA true
LTAI true
HBOR 50
TBOR 100
HCON FBD
TCON FBD
DETA true
LNTP unset
TEMP -100000
PRES 0
HSTU /A3B/PA50
CCEN 0
CCLA 0
PSPE /A3B
DUTY unset
DSCO unset
PTSP unset
INSC unset
PTNB 0
DELDSG unset

END
```

Output Syntax for Nozzles and Branches

```
OUTPUT ONLY (NOZZ BRAN) /HTEST
```

Output

```
NEW NOZZLE /VENT_OUT
UCOFG E 0 N 0 U 0
POS E 0 S 500 U 1600
ORI Y is U and Z is S
TEMP -100000
PRES 0
HEIG 200
DUTY unset
DESP 400 200 25 5

END
NEW BRANCH /100-B-1-B1
BUIL false
SHOP false
HPOS E 12490 N 12280 U 1150
TPOS E 4979 N 9887 U 13655
HDIR U
TDIR N
UCOFG E 0 N 0 U 0
LHEA true
LTAI true
HBOR 50
TBOR 100
HCON FBD
TCON FBD
DETA true
LNTP unset
TEMP -100000
PRES 0
HSTU /A3B/PA50
CCEN 0
CCLA 0
PSPE /A3B
DUTY unset
DSCO unset
PTSP unset
INSC unset
PTNB 0
DELDSG unset

END
```

13.4.9 PASS Option

Optionally output 1st pass or 2nd pass only

Output Syntax 1st Pass Only

```
OUTPUT PASS 1 /EQUIP
```

Output

```
INPUT BEGIN
NEW ZONE /EQUIP
FUNC 'Equipment - above grade'
PURP EQUI

NEW EQUIPMENT /E1301
POS E 2850 N 5660 U 1470
UCOFG E 0 N 0 U 0
BUIL true
DSCO unset
PTSP unset
INSC unset

NEW CYLINDER
POS E 0 N 3299 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 960
HEIG 6598
```

Output Syntax 2nd Pass Only

```
OUTPUT PASS 2 /EQUIP
```

Output

```
INPUT BEGIN
OLD /E1301-S1
CREF /200-B-4-B1  TAIL
CATR /AAZFBD0TT

OLD /E1301-S2
CREF /250-B-5-B1  HEAD
CATR /AAZFBD0TT

OLD /E1301-S3
CREF /250-B-5-B2  HEAD
CATR /AAZFBD0TT
```

```
OLD /E1301-T1
CREF /100-C-12-B2  TAIL
CATR /AAZFBB0NN

OLD /E1301-T2
CREF /100-C-13-B1  HEAD
CATR /AAZFBB0NN
```

13.4.10 Option Combinations

It is possible to use all of the above options in an output syntax to achieve a combination of results. The following commands would be valid:

```
OUTPUT COMMENT TABULATE 4 INDEX /RACKPIPES
OUTPUT COMMENT TABULATE 2 INDEX BRIEF /ELECT
OUTPUT ONLY (ZONE NOZZ BRAN) INDEX TABULATE 2 /HTEST
OUTPUT OLDFORMAT INDEX TABULATE 4 NOUDA /HEATING-VENTS
OUTPUT INDEX COMMENT PASS 1 /CABLETRAY
```

13.4.11 Locate and Replace

The following Locate and Replace syntax is possible

Output Syntax Locate/Replace

```
OUTPUT LOCATE REPLACE /E1301
```

Output

```
INPUT BEGIN
NEW LOCATE SITE /HTEST
NEW LOCATE ZONE /EQUIP
NEW REPLACE EQUIPMENT /E1301
POS E 2850 N 5660 U 1470
UCOFG E 0 N 0 U 0
BUIL true
DSCO unset
PTSP unset
INSC unset

NEW CYLINDER
POS E 0 N 3299 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 960
HEIG 6598

END
```

```
NEW CYLINDER
POS E 0 N 6848 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 1020
HEIG 500

END
```

Output Syntax Replace Only

```
OUTPUT REPLACE /E1301
```

Output

```
INPUT BEGIN
NEW REPLACE EQUIPMENT /E1301
POS E 2850 N 5660 U 1470
UCOFG E 0 N 0 U 0
BUIL true
DSCO unset
PTSP unset
INSC unset

NEW CYLINDER
POS E 0 N 3299 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 960
HEIG 6598

END
NEW CYLINDER
POS E 0 N 6848 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 1020
HEIG 500

END
NEW BOX
POS E 0 N 1710 D 315
LEVE 0 8
XLEN 460
YLEN 300
ZLEN 630
```

```
END
```

Output Syntax Locate Only

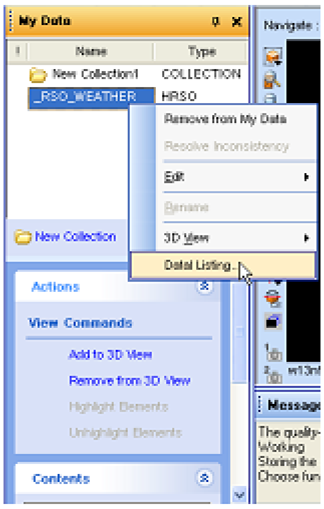
```
OUTPUT LOCATE /E1301
```

Output

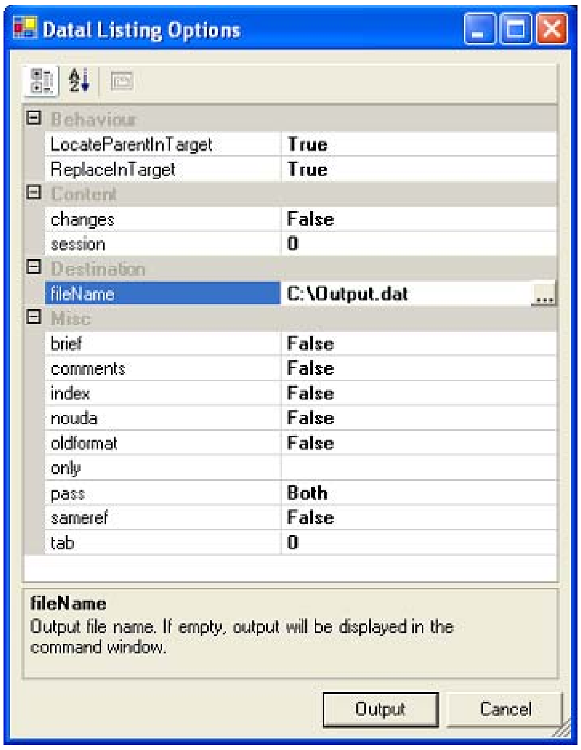
```
INPUT BEGIN
NEW LOCATE SITE /HTEST
NEW LOCATE ZONE /EQUIP
NEW EQUIPMENT /E1301
POS E 2850 N 5660 U 1470
UCOFG E 0 N 0 U 0
BUIL true
DSCO unset
PTSP unset
INSC unset

NEW CYLINDER
POS E 0 N 3299 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 960
HEIG 6598

END
NEW CYLINDER
POS E 0 N 6848 U 0
ORI Y is D and Z is N
LEVE 0 4
DIAM 1020
HEIG 500

END
NEW BOX
POS E 0 N 1710 D 315
LEVE 0 8
XLEN 460
YLEN 300
ZLEN 630

END
```

13.4.12 Create Datal Interactively

In the MyData window, the **Datal Listing** menu item on the context menu can be used to create a Datal file.



This will display the **Datal Listing Options** form, where different options can be set. If the file name is left empty, the Datal will be written to the Command Window.


Datal Listing Options

Behaviour	
LocateParentInTarget	True
ReplaceInTarget	True
Content	
changes	False
session	0
Destination	
fileName	C:\Output.dat
Misc	
brief	False
comments	False
index	False
nouda	False
oldformat	False
only	
pass	Both
sameref	False
tab	0

fileName
Output file name. If empty, output will be displayed in the command window.

14 Project Queries

14.1 MDB Mode

The MDB command puts you into **MDB Mode**, where you can use a limited number of Monitor commands. This lets you change the current multiple database during a DESIGN session without having to leave the Module and enter Monitor.

When you enter MDB mode, you can either update the current MDB to save your design changes, or ignore any changes made since your last SAVEWORK command, by specifying UPDATE or NOUPDATE.

When you are in MDB mode, you can give the following commands, which are the same as the corresponding Monitor commands. For more information, see the *Monitor Reference Manual*.

Examples:

`MDB UPDATE`	Save design changes and enter MDB Mode.
`MDB NOUPDATE`	Enter MDB Mode without saving design changes.

`EXCHANGE`
`DEFER`
`CURRENT`
	Alter the databases in the current list of the current MDB

`PROTECT`	Temporarily alters your access rights to specified databases.
`USER` *`username`*	Changes the current user
`PROJECT` *`code`*	Changes the current project
`LIST`	Allows you to query: Users, including the number of active users, Teams including the set (current) Team, Databases, including copied Databases, MDBs, Macros and Variables
`/PIPING`	Change to MDB /PIPING.
`/PIPING READONLY`	Change to MDB /PIPING in read-only mode.
`EXIT`	Return to DESIGN Mode.

Command Syntax:

```
>-- MDB --+-- UPdate ----.
          |              |
          '-- NOUPdate --+-->
```

14.2 Checking the Current User Status

Gives you information about your current status as a user and about the DBs to which you have access.

Example:

A typical response to the STATUS command could be:

```
Project: XYZ
User:    CSI (758)
Teams:   B
MDB:     /DESIGN
Current DBS:
  1  PIPING/SITE                  RW
  2  MASTER/CATLOG                R
Deferred DBS:
  3  STRUCT/STEEL
```

This indicates that the designer has identified himself as being Outfitting user CSI, that he is logged in to the computer as user 758, that he is a member of team B, that he is accessing Project XYZ, and that he has selected an MDB called /DESIGN.

Command Syntax:

```
>-- STATus -->
```

14.3 Checking the Current System Status

The **SY**stem **STAT**us command gives you information about the current active status of the project in which you are working. It lists all users who are currently accessing the project, the modules and databases which they are using, and whether they are examining (Read-only status) or modifying (Read/Write status) the database. It also gives the workstation identifier for each user.

Example:

A typical response to the SYSTAT command could be:

```
PROJECT  XYZ
==============

USER SYSTEM (57b)
MODULE ADMIN
MDB ** UNSET **

USER HHJ (752)
MODULE DESIGN
MDB /STEEL

DB              MODE
MASTER/AREA-A   R
MASTER/AREA-B   R
STRUC/AREA-C    RW
```

This shows that two users are currently logged in and are using Outfitting for work on Project XYZ. The Project Coordinator is using ADMIN but is not accessing any databases. User 752 is using DESIGN. He is accessing the MDB named /STEEL, whose constituent DBs are as listed. He has Read-only status for the DBs owned by the MASTER (System) team and Read/Write access to the DB STRUC/AREA-C.

Command Syntax:

```
>-- SYStat -->
```

14.4 Listing Project Information

Lets you list most of the project information held in the System Database, with the exception of confidential details such as other users' passwords, which can only be listed by the System Administrator in Outfitting ADMIN.

Examples:

A typical response to the LIST MDB command could be:

```
 List of MDBS for project XXX
  ==============================
  MDB:     /DESIGN
  Current DBS:
    1  PIPING/AREA-A                    DESI Exclusive
    2  PIPING/AREA-C                    DESI Update
    3  MASTER/AREA-D                    DESI Exclusive
```

Examples:

```
 Deferred DBS:
    4  PIPING/AREA-B                       DESI Exclusive
    5  MASTER/AREA-E                       DESI Update

  MDB:/STEEL
  Current DBS:
    1  MASTER/AREA-A                       DESI Exclusive
    2  MASTER/AREA-B                       DESI Exclusive
    3  STRUCT/AREA-C                       DESI Exclusive
  Deferred DBS:
  **NONE**

  MDB:     /ANSI
  Current DBS:
    1  CATAL/AREA-E                        CATA Update
  Deferred DBS:
  **NONE**
```

A typical response to the LIST USERS command could be:

```
 List of USERS for project ZZZ
  ===============================
  SYSTEM   (FREE)
  TEAMS :MASTER   STAB

  Z        (FREE)
  TEAMS :***NONE**

  GEN      (GENERAL)
  TEAMS :TEST
```

The information generated by the LIST command can either be displayed on screen or sent to a file.

Command Syntax:

```
             .----<----.
            /          |
>-- LIst --*-- USers --|
           |           |
           |-- MDBs ---|
           |           |
           |-- DBs ----|
           |           |
           |-- TEams --'
           |
           '-------------->
```

14.5 Querying MDB Information

Lets you query details of the current MDB. This is a supplementary querying facility to the LIST command (*Listing Project Information*). It allows specific information about features of the project configuration to be interrogated.

Command Syntax:

```
>-- Query --+-- USer      ---.
            |                |
            |-- USer word ---|
            |                |
            |-- TEam word ---|
            |                |
            |-- DB dbname ---|
            |                |
            ‘-- MDB name ----+-->
```

14.5.1 Querying Individual Database Information

Lets you query details of the current DB for a given element.

Examples:

`Q DBNAME`	Gives name of current DB
`Q DBTYPE`	Gives type of current DB
`Q DBFNUMBER`	Gives file number for current DB
`Q DBFILE`	Gives pathname for current DB file

Command Syntax:

```
>-- Query --+-- DBNAme -----.
            |               |
            |-- DBTYpe -----|
            |               |
            |-- DBFNumber --|
            |               |
            ‘-- DBFIle -----+-->
```

15 Link Documents

The link documents mechanisms provide the ability to link external and internal documents to database objects. Every element in the database can be associated with a resource that is either another database element for example a drawing, or an external document such as a file, or a web page. External documents are not stored in the Dabacon database.

15.1 Overview

Each document or other resource, either external or internal, that can be linked to a database element is represented in the database as Link Descriptor. The Link Descriptor's main role is to carry information about the document it describes and a Uniform Resource Locator (URL).

It is possible for any other elements in the database to reference these Link Descriptors through a two-way mechanism enabling users to find all elements that reference a particular Link Descriptor and the opposite, find all documents referenced by an element.

It is also possible to assign classification information to each Link Descriptor. The classification information provides the facility of assigning multiple class information to a Link Descriptor so that a search for all elements that have references to documents with specific classification assigned can be made. For example, a search can be made for all Link Descriptors classified as "Installation"-class document or all pumps that do not reference any "Certificate" and "Security"-class documents.

The following figure shows a schematic overview of the possible linkage to external documents and internal drawings.

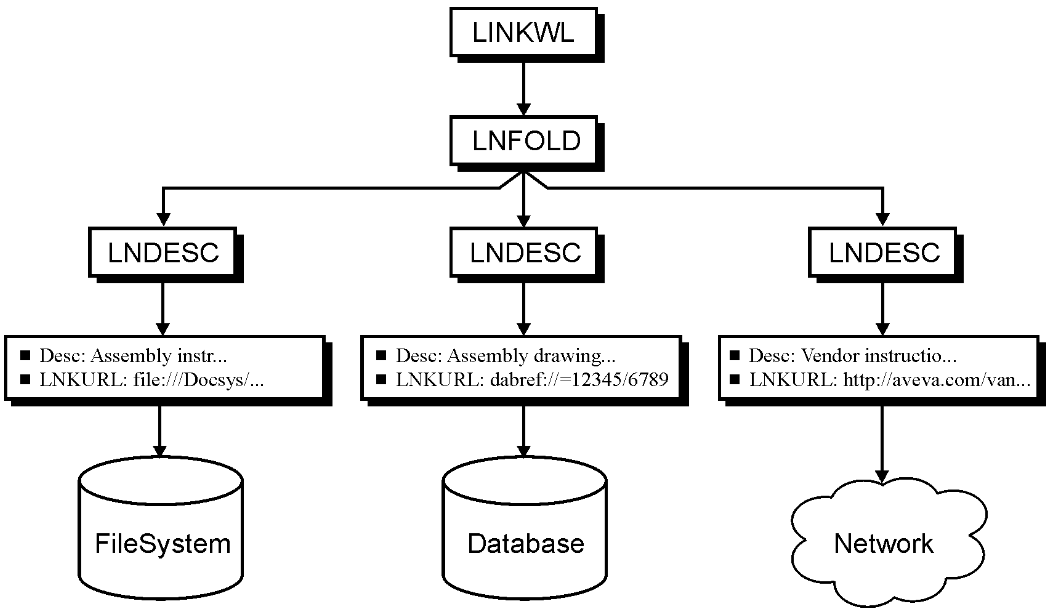

LINKWL
LNFOLD
LNDESC
LNDESC
LNDESC
■ Desc: Assembly instr...
■ LNKURL: file:///Docsys/...
■ Desc: Assembly drawing...
■ LNKURL: dabref://=12345/6789
■ Desc: Vendor instructio...
■ LNKURL: http://aveva.com/van...
FileSystem
Database
Network

Figure 15:1. Examples of references to internal and external documents

15.2 Data Structures

There are several element types used for organizing links and storing link information. All kinds of elements may be created using standard Outfitting command syntax.

15.2.1 Link World (LINKWL)

All elements related to links are stored under Link World elements. To use links you have to create at least one Link World. It can store Link Folders, Link Classes and Link Descriptors which are covered in the following sections.

It is possible to assign Link Descriptors to elements in other databases. It is therefore recommended that LINKWL elements are created in a DESIGN database of its own to which all relevant teams have read and write access.

15.2.2 Link Folder (LNFOLD)

Under Link Worlds it is possible to organise all elements into a tree structure. You can create Link Folders that can contain more Link Folders or Link Classes and Link Descriptors. This way it is possible to freely configure the hierarchy.

15.2.3 Link Descriptor (LNDESC)

A Link Descriptor (LNDESC) element holds a link to documents and external resources. Both external documents and draft drawing elements can be referenced using a LNDESC.

A Link Descriptor has the following attributes:

- NAME - user-defined name of the LNDESC element.
- DESC - description of the element.

- LNKURL - a string storing raw Uniform Resource Locator of the linked document.

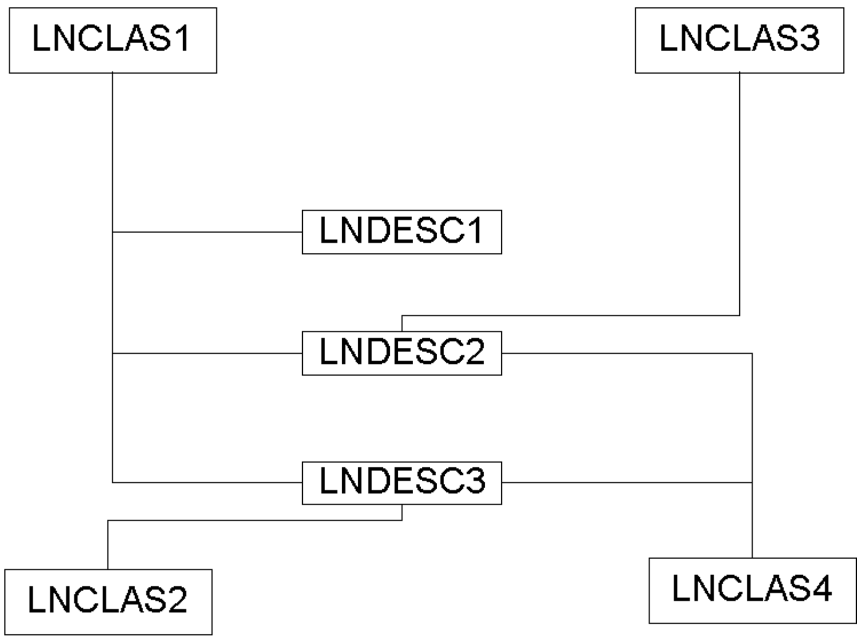
  It can be for example a file ("file:///Docsys/ProjectX/MyDocument.doc"), a web page ("http://www.aveva.com"), an e-mail address ("mailto:support@aveva.com") or any other external resource. If it is an internal database reference (e.g. to a drawing) it is stored in a form "dabref://=12345/6789".

The following pseudo attributes may be queried:

- LNKREF - if the Link Descriptor holds a reference to an internal database element, i.e. the LNKURL stores a "dabref://…" link, this attribute returns a reference to a database element linked through this descriptor. Otherwise, LNKREF returns a null reference. It is also possible to store an internal reference through this attribute.
- URL - this attribute returns merely the value of LNKURL but its main purpose is that you can assign either an external URL or a string with reference number of a Dabacon element, which is automatically recognised and stored as an internal link.
- LNKCLS - list of Link Class elements that classify this Link Descriptor.
- LNKELE - list of elements that have this Link Descriptor assigned.

15.2.4 Link Class (LNCLAS)

Classification of documents is possible through use of Link Classes (LNCLAS). Each Link Descriptor (LNDESC) may be classified by multiple classes; in the diagram below see how each LNDESC is associated with more than one LNCLAS. A single Link Class may classify multiple Link Descriptors; in the diagram LNCLASI is associated with all three LNDESC.



A Link Class has the following attributes:

- NAME - user-defined name of the LNCLAS element.
- DESC - description of the element.

There is also a pseudo attribute available named LNKDOC that returns a list of LNDESC elements that are classified by this LNCLAS.

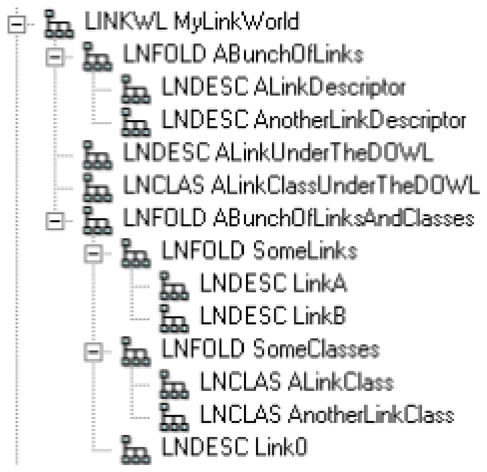
15.3 Command Line User Interface

Link Documents functionality is available through the command line. Standard Outfitting command syntax can be used to create, delete or modify database elements and to query and set their attributes including pseudo attributes.

The following sections describe the typical scenarios connected with links including examples.

15.3.1 Configuring Links Hierarchy

Before it is possible to link documents to database elements it is necessary to create at least one Link World (LINKWL). You can organize Link Descriptors and Link Classes into a hierarchy of folders. An example hierarchy is shown below.



```
LINKWL MyLinkWorld
    LNFOLD ABunchOfLinks
        LNDESC ALinkDescriptor
        LNDESC AnotherLinkDescriptor
    LNDESC ALinkUnderTheDOWL
    LNCLAS ALinkClassUnderTheDOWL
    LNFOLD ABunchOfLinksAndClasses
        LNFOLD SomeLinks
            LNDESC LinkA
            LNDESC LinkB
        LNFOLD SomeClasses
            LNCLAS ALinkClass
            LNCLAS AnotherLinkClass
        LNDESC Link0
```

Figure 15:2. An example hierarchy under a Link World

15.3.2 Linking a Document to a Database Element

To add a link between an element and a document (or an external resource):

1. Create a new Link Descriptor somewhere in the links hierarchy.
2. Set its URL to the desired internal or external reference.
3. Link it to a database element using a DLADD command.

You can also link a document to an element using an existing Link Descriptor, because many database element can be linked to the same document:

1. Locate a LNDESC in the links hierarchy.
2. Link it to a database element using a DLADD command.

The DLADD command can be used to add links from a document to one or more other elements. The command syntax is:

```
                        ----------------+
                       /                |
>--- DLADD -------*-- <selatt> -----+------>
```

It is possible to create an association both by adding a link from a Link Descriptor to a database element and by adding a link from a database element to a Link Descriptor. Examples are given below.

If current element is a Hull Panel Element (HPAN) named /PANEL1 the following command assigns Link Descriptors /MYDOC1 and /MYDOC2 to /PANEL1:

```
> DLADD /MYDOC1 /MYDOC2
```

If current element is a Link Descriptor (LNDESC) named /MYDOC1 the following command assigns this Link Descriptor to /PANEL1 and /PUMP1:

```
> DLADD /PANEL1 /PUMP1
```

The whole process of linking a document to /PUMP1 might look like this:

```
> NEW LNDESC /MYDOC
> URL 'http://aveva.com/all_about_vm12_link_documents.pdf'
> DLADD /PUMP1
```

15.3.3 Unlinking a Document from a Database Element

Once there is an association between an element and a Link Descriptor you can break the assignment by using the DLREMOVE command. The command syntax is:

```
                    ----------------+
                   /                |
>--- DLREMove ----*-- <selatt> -----+------>
```

It is possible to remove an association both by removing a link from a Link Descriptor to a database element or by removing a link from a database element to a Link Descriptor. For example:

If current element is a Hull Panel Element (HPAN) named /PANEL1 the following command removes link to document described by /MYDOC1 from /PANEL1:

```
> DLREM /MYDOC1
```

If current element is a Link Descriptor (LNDESC) named /MYDOC1 the following command removes link to document described by this Link Descriptor to /PUMP1:

```
> DLREM /PUMP1
```

The following command removes all Link Descriptor associations from the current element:

```
> DLREM LINks
```

The following command removes the first five Link Descriptor associations from the current element:

```
> DLREM LIN 1 TO 5
```

Getting and setting link target

If the current element is a Link Descriptor you can retrieve or set the URL stored in this descriptor. To link to an external resource you should directly set the URL:

```
> URL 'file:///Docsys/MyDocument.doc'
```

The Link Descriptor has a pseudo attribute LNKREF that returns a database reference if the descriptor links internally to Dabacon. If you set the URL to an external resource LNKREF returns a null reference:

```
> QUERY LNKREF
```

Url DBRef Nulref

You can use the LNKREF to set link to an internal database reference e.g. a drawing:

```
> LNKREF /DRAWING1
> QUERY LNKREF
Url DBRef / DRAWING1
```

15.3.4 Classifying a link

Each Link Descriptor can have a number of Link Classes assigned. To classify a link:

1. Create a Link Class element (LNCLAS) somewhere in the links hierarchy.
2. Assign the class to the Link Descriptor with a DLADD command.

If current element is a Link Descriptor (LNDESC) named /MYDOC1 the following command classifies this Link Descriptor as a /MYCLASS1 and /MYCLASS2 document:

```
> DLADD /MYCLASS1 /MYCLASS2
```

If current element is a Link Class (LNCLAS) named /MYCLASS1 the following command classifies a Link Descriptor named /MYDOC1 as a /MYCLASS1 document:

```
> DLADD /MYDOC1
```

15.3.5 Unclassifying a link

To remove classification information from a Link Descriptor you can use the DLREMOVE command.

If current element is a Link Descriptor (LNDESC) named /MYDOC1 the following command removes /MYCLASS1 classification from the /MYDOC1 Link Descriptor:

```
> DLREM /MYCLASS1
```

If current element is a Link Class (LNCLAS) named /MYCLASS1 the following command removes /MYCLASS1 classification from the /MYDOC1 Link Descriptor:

```
> DLREM /MYDOC1
```

The following command removes all Link Descriptor associations from the current Link Class:

```
> DLREM LINks
```

The following command removes all classification information from the current Link Descriptor:

```
> DLREM CLAsses
```

15.3.6 Related Pseudo Attributes

A number of pseudo attributes allow retrieval of information on the relation between Link Descriptors, Link Classes and model elements:

1. For each element in a database you can query the LNKDOC pseudo attribute to get a list of all Link Descriptors (e.g. documents) that are associated with this element.
2. For each Link Descriptor by querying the LNKELE pseudo attribute you get a list of all database elements linked to the document associated with this descriptor.
3. For each Link Descriptor you can query the LNKCLS pseudo attribute to get a list of all classes that classify the document associated with this descriptor.
4. For each Link Class by querying the LNKDOC pseudo attribute you get a list of all Link Descriptors that have this Link Class assigned as classification information.

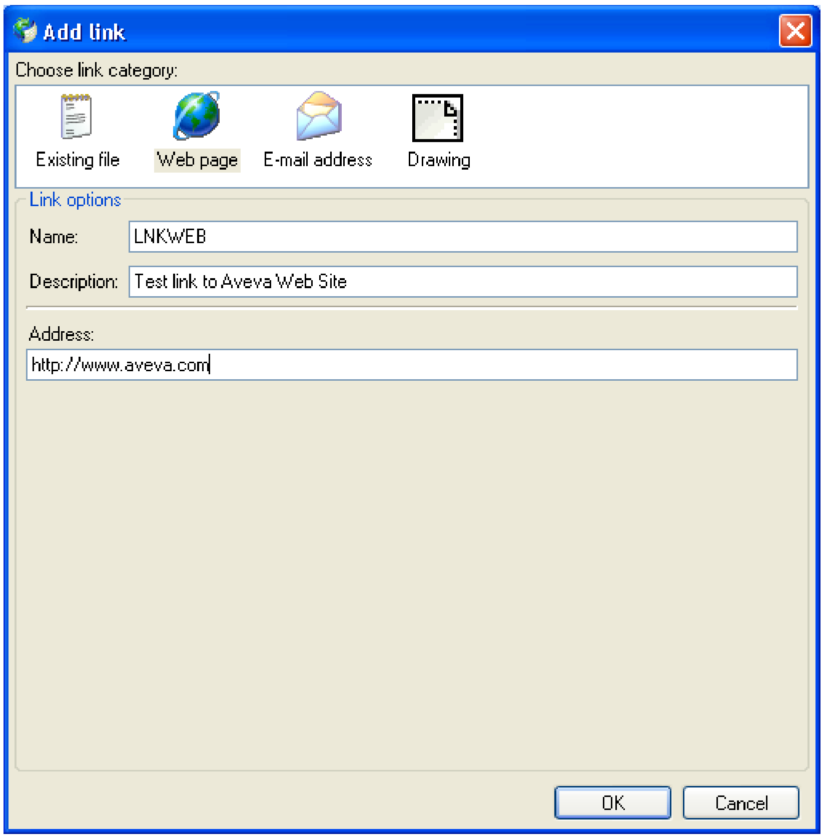
15.4 Links Addin

The Link Addin is a customisable user form which simplifies much of the process of creating links.

The Links Addin uses the notion of link categories to treat different types of links differently.

By default the Links Addin comes with predefined link categories for: E-mail address, Web page, Existing file and Drawing. However, it gives the possibility to extend this set of link types and to create additional categories. For example, one could add a category that would accumulate links to documents or links to FTP resources if needed.

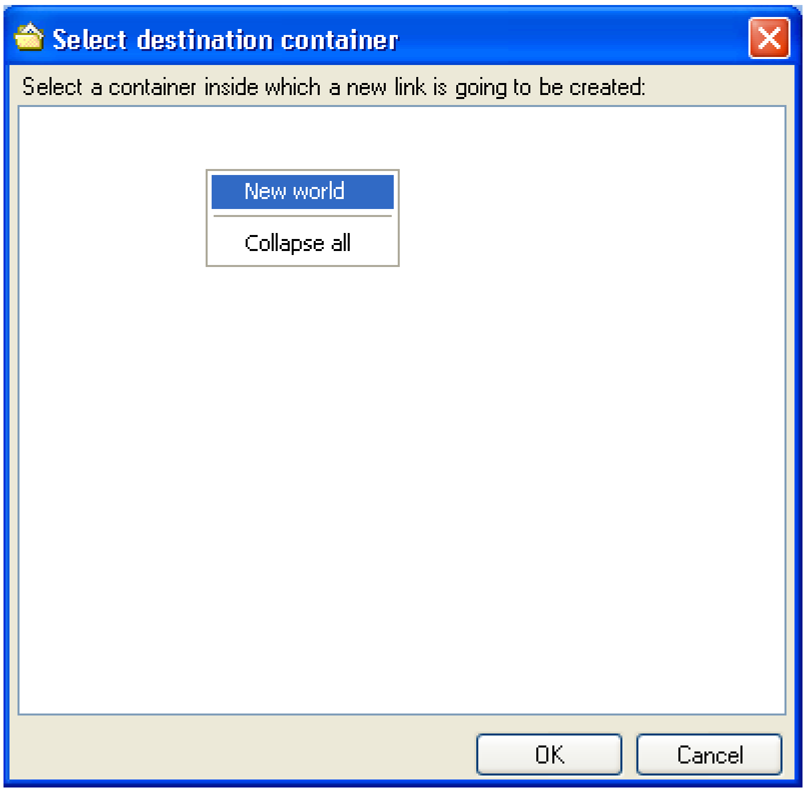
When creating a link the Links Addin gives the possibility to choose a link category and set options for the new link. An example link creation dialog is shown below.



Name: LNKWEB
Description: Test link to Aveva Web Site
Address:
http://www.aveva.com

Each link category has a name and an icon. The dialog provides the possibility to input the Name and Description for the link, depending on the type of link being created the form will prompt for an appropriate resource to link to. For example when created a link to a web page the user is prompted to enter a valid Address such as http://www.aveva.com.

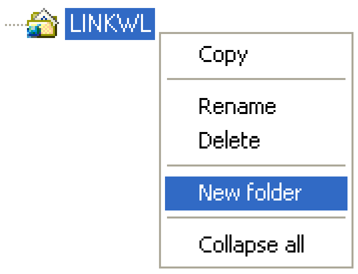
Clicking OK will open a new window prompting for a container for the new link.


Select destination container
Select a container inside which a new link is going to be created:
New world
Collapse all
OK
Cancel

On its first use, the ‘Select destination container’ window will appear empty. This is because a Link World and Link Folder hierarchy has not been created.

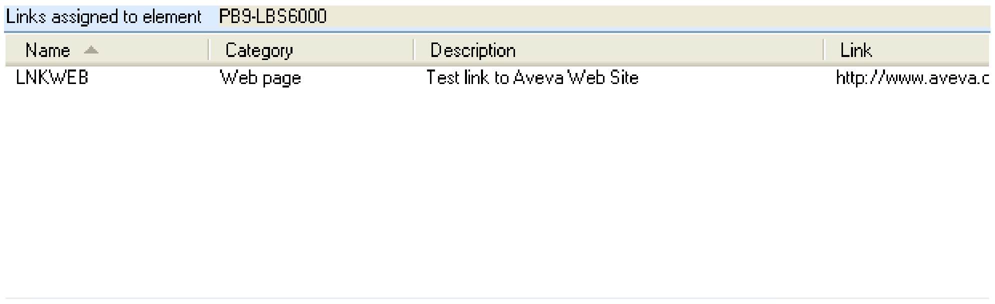
As discussed in the section ‘Configuring Link Hierarchy’ at least one Link World should be created.

In the ‘Select destination container’ window right click to create a ‘New world’. A ‘New folder’ must also be created below a world.



Click OK for the link to be created in the database hierarchy.

Once a link has been created it is possible to view the link attributes using the link list sub form, launched from the ‘Show Link’ button from the toolbar. This form displays the element the Link has been assigned to, the Link Name, Category, Description and Link URL.



Links assigned to element PB9-LBS6000

Name	Category	Description	Link
LNKWEB	Web page	Test link to Aveva Web Site	http://www.aveva.c

The Link Addin can be customised through a set of API's, for further information refer to the *Software Customisation Reference Manual*.

16 Inter-DB Connection Macros

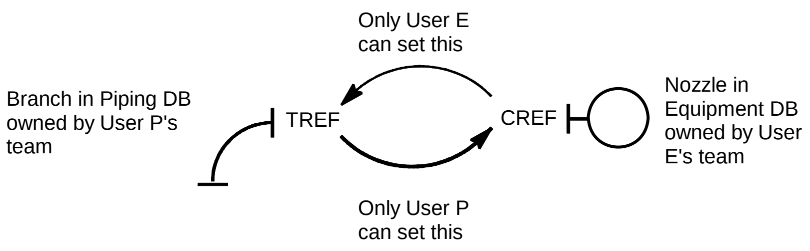
Access to a DB is usually controlled in such a way that only *one* user can modify the content of that DB at any one time; that is, only one user can have **Write** access to the DB. Other users may have simultaneous **Read** access, depending how access rights have been set up in the ADMIN module.

In a multi-disciplinary project, in which different teams of users work on different aspects of the design, an individual user will usually have Read/Write access to the DBs controlled by their own team and Read-only access to DBs controlled by other teams. This works well until a user needs to connect an item in their team's DB to an item in another team's DB; for example, a member of the Piping team may wish to connect a Branch in a Piping DB to a Nozzle in an Equipment DB (to which they have Read-only access). In such a case, the design changes needed in the Equipment DB are stored in a 'buffer' file known as an **inter-DB connection macro**. Only when this macro is run by a member of the Equipment team, with Write access to the Equipment DB, are the changes implemented.

The sequence of events which would occur is illustrated in the following example.

Assume that **Project ABC** has separate Piping and Equipment design teams. Assume that **User P** has Read/Write access to a Piping DB and Read-only access to an Equipment DB, while **User E** has Read/Write access to the Equipment DB and Read-only access to the Piping DB.

User P wishes to connect a Branch Tail in their Piping DB to a Nozzle in User E's Equipment DB; that is, they wish to set the Branch's TREF in their Piping DB to point to the CREF of the Nozzle (which they can do) and to set the CREF of the Nozzle to point to the TREF of their Branch (which they can *not* do), thus:


Only User E can set this
Branch in Piping DB owned by User P's team
TREF
CREF
Nozzle in Equipment DB owned by User E's team
Only User P can set this

- User P sets the TREF of their Branch to point to the CREF of the Nozzle in the Equipment DB.
- When User P tries to set the Nozzle's CREF, they receive a message telling them that they are trying to connect to a read-only DB and that an inter-DB connection macro is being created automatically. This macro, which stores the commands needed to set the CREF, is given a name with the format **abc001.mac** (where the macro number, 001 here, is allocated sequentially), and is held in the directory **ABCMAC** (or as defined by the project's environmental variables).

- When User E next enters MONITOR, they receive a message asking them to run the inter-DB connection macro abc001.mac and to delete it when they have done so.
- User E enters DESIGN and runs the inter-DB connection macro by giving the command

  ```
  $M /%ABCMAC%/abc001.mac
  ```

  This sets the CREF for the Nozzle to point to the TREF of the Branch and completes the link between the two DBs.
- User E enters MONITOR (or ADMIN if they have sufficient access rights) and deletes the redundant macro by giving the command

  ```
  DELETE MACRO 1
  ```

  where 1 is the macro number. This command is valid in DESIGN, MONITOR and ADMIN.

Note: If User P checks their DB for data consistency errors between Stages 2 and 4, when the macro has been created but not yet run, they will get an ‘incompatible connection reference’ message. They cannot finalise their design until User E has run the macro. Thus, the successful use of inter-DB connection macros relies on good co-operation between the teams involved.

Note: Inter-DB connection macros are also created in multiwrite DBs if an attachment is claimed by another user.

17 Automatically Prompting the Save Dialogue

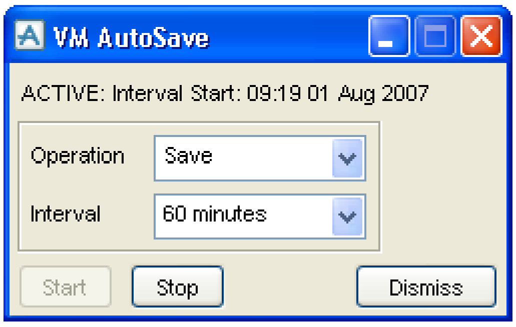
The Autosave utility in the DESIGN module prompts the user to save their work at regular intervals, which can be determined by the user. The utility will prompt the user to confirm a Save operation so that its effect on the current session can be managed.

In the **DESIGN - General Application**, the Autosave function is configured by selecting:

Settings > Save Work Options

Note: This does not effect the Save Work command itself accessed by selecting **DESIGN > Save Work**.

If the **Save Work Options...** command does not appear in the **Settings** menu, then this utility has not been installed, and the system will not prompt the user to save work at regular intervals. Otherwise the **AutoSave** form displays:



ACTIVE: Interval Start: 09:19 01 Aug 2007

Operation: Save

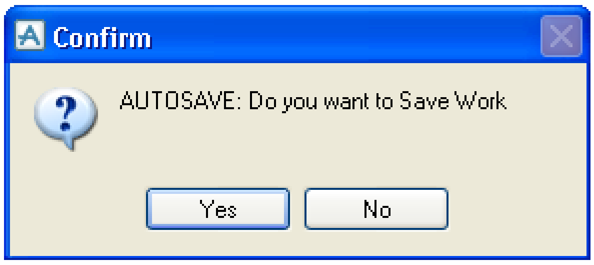
Interval: 60 minutes

The **AutoSave** form provides the following functions:

Operation	Sets the operation to be performed:	
	Save	Savework command only
	Save/Getwork	Savework and Getwork commands
Interval	Sets the interval between reminders to save work to 15, 30, 45 or 60 minutes	
Start	Starts the service	
Stop	Suspends the service	
Dismiss	Removes the form without starting or stopping the service	
Active	Shows the start time and date of the current interval.	

An interval starts when the user performs a Save Work using a the GUI, or when the service was last started.

In normal operation, users will be prompted by a **Confirm** form, if they do not save work within the time interval specified on the **AutoSave** form. Note that answering **No** to this forms starts a new time interval even though work has not been saved.



Notes:

1. Each Save Work creates a new session on the database, which increases the size of the database.
2. The "undo" command only operates between Save Work commands. Once a Save Work command is performed, it is not possible to undo operations performed prior to the last Save Work.
3. Data is permanently saved to the database when a Save Work command has been performed. The user cannot remove changes saved to the database. A Project Administrator can remove changes by rolling back sessions in the Administrator module.
4. The user may experience a brief delay while the Save Work command performs its task.

Installation

This utility is installed using the PML Add-ins mechanism described in the *Software Customisation Guide*.

This add-in is specified by add-in definition file with pathname %PDMSUI%\DES\ADDINS\asave. This add-in file is delivered with the product, but it must be edited to enable the utility.

In order to do this, open the **asave** file with a text editor. It should contain the following commented add-in instructions.

```
# ----------------------------------------------------------------------
#
#  (c) Copyright  2007 to Current Year  AVEVA Solutions Limited
#
#  File:             ASAVE
#  Type:             Add-in Definition
#  Module:           design
#
#  Author:           Malcolm Barlow
#  Created:          Wed Mar 28 2007
#
#  Description:      Add-in definition for autosave application
#
```

```
# --------------------------------------------------------------------------
#
# Name:           ASAV
# Directory:      GEN
# Title:          Autosave
# Object:         apphsave
# ShowOnMenu:     FALSE
# ModuleStartup:!!appHsave.initialise(true)
# StartupModify: GEN    :!!appHsave.modifyMenus()
```

Remove the # character from the lines containing add-in instructions. Leave the # character at the beginning of comment lines. The resulting file should appear as follows:

```
# --------------------------------------------------------------------------
#
#  (c) Copyright  2007 to Current Year  AVEVA Solutions Limited
#
#  File:             ASAVE
#  Type:             Add-in Definition
#  Module:           design
#
#  Author:           Malcolm Barlow
#  Created:          Wed Mar 28 2007
#
#  Description:      Add-in definition for autosave application
#
# --------------------------------------------------------------------------
#
  Name:           ASAV
  Directory:      GEN
  Title:          Autosave
  Object:         apphsave
  ShowOnMenu:     FALSE
  ModuleStartup:!!appHsave.initialise(true)
  StartupModify: GEN    :!!appHsave.modifyMenus()
```

Save the changes to the **asave** file. Restart for these changes to take effect.

By changing the **asave** file it is possible to configure the system to start with autosave enabled or disabled. The file as shown above starts DESIGN with autosave enabled. In order to start the system with autosave installed and disabled, change the following line:

```
    ModuleStartup:!!appHsave.initialise(true)
```

to

```
    ModuleStartup:!!appHsave.initialise(false)
```

18 Sequence Number Generator

This is a guide for PML application programmers on how to handle the common problem of deriving unique names in a concurrent application environment. The functionality is based on a set of methods that operates towards a separate overwrite type of database. This is storing the uniquely named sequences and also details about the respective sequence. Name sequence elements and their attributes are protected from being handled manually. One way to operate with name sequences is through PML NameSeq objects.

A unique item in a sequence is made up by the name of the sequence followed by a running number, e.g. TEST123, where TEST is the name of the sequence and 123 is the running number.

18.1 Create a Name Sequence Database

A name sequence database, of the type NSEQ, is created by the Admin utility and selected to an MDB as any other ordinary type of database. The options given while creating this type of database is limited as the NSEQ type of the database is predestined to be of an overwrite type.

18.2 Enable Usage of Name Sequences from PML

import '`aveva.pdms.nameseq`'

using namespace '`aveva.pdms.nameseq`'

18.3 NameSeq Object

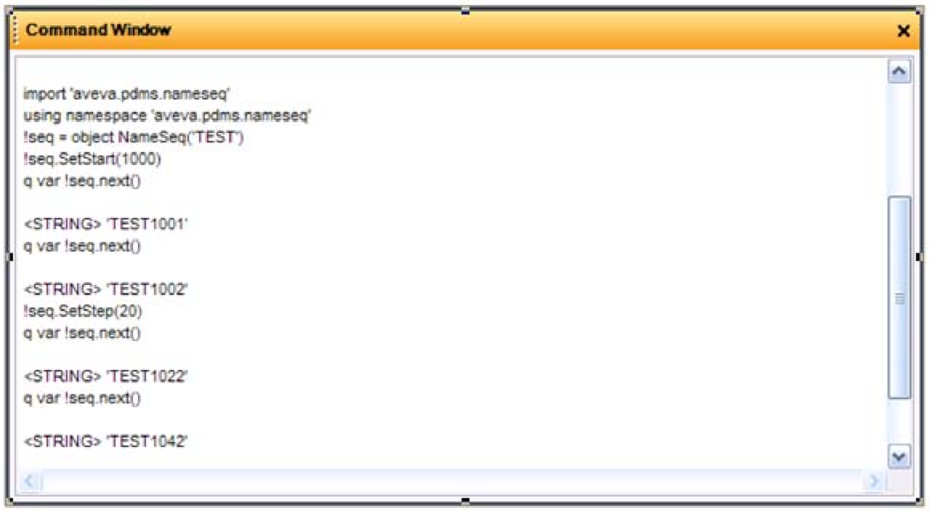
Methods

Name	**Result**	**Description**
NameSeq(string)	Bool	If not existing, creates a sequence of given name, otherwise brings back the name sequence available.
Next()	String	Get next composed name in sequence.
Remove()	Bool	Delete sequence.
SetStart(real)	Bool	Set first running number of sequence (default 0).
SetMax(real)	Bool	Set last running number of sequence.
SetStep(real)	Bool	Set increment (default 1).

SetWraparound()	Bool	Allow wraparound when maximum value is reached.
SetNoWraparound()	Bool	Disallow wraparound (error returned when maximum is reached).
GetMax()	Real	Get last running number of sequence (default 2147483647).
GetStep()	Real	Get increment.
GetCurrent()	Real	Get current running number.
GetName()	String	Get name of sequence.
IsWraparound()	Bool	Get wraparound status.

18.4 Typical Usage of Name Sequences

Following is an example showing how to define a sequence named TEST and let the sequence start from 1000. The method **Next** increments the running number and returns a name composed by the name of the sequence followed by the running number, i.e. TEST1001, TEST1002. Whenever needed it is possible to make a break-in in the sequence and change for example the increment to e.g. 20 instead of 1 as default. This will then generate composed names as TEST1022, TEST1042.



```
import 'aveva.pdms.nameseq'
using namespace 'aveva.pdms.nameseq'
!seq = object NameSeq('TEST')
!seq.SetStart(1000)
q var !seq.next()

<STRING> 'TEST1001'
q var !seq.next()

<STRING> 'TEST1002'
!seq.SetStep(20)
q var !seq.next()

<STRING> 'TEST1022'
q var !seq.next()

<STRING> 'TEST1042'
```

18.5 Name Sequences in Global Projects

Name sequence databases are local for each site and are not replicated among the different sites. However names generated by the name sequence mechanism can be used for naming of items included in the Global synchronisation. To maintain unique naming of items in a Global environment, the setup of name sequences must be considered. A proper method is to let each site have different starting numbers of their name sequences, i.e. SetStart() to a number different for each site.