Single line commands

Important characters

#	Precedes a comment anywhere on a line
&	Line continuation
;	Line separator e.g. cmd1;cmd2

Keyword Commands

Form *command/qualifiers parameters*
e.g. display/line/errors w1 0 0 100 100.
The '_' character is used to represent undefined parameters e.g. display/line w1 _ _ 100 100.

Assignments

These take the form <*destination*>=<*expression*>, such as i=6.

Expressions

literal values	"Hi",4.5
variables	x,theta
bracketed expressions	(x-7)
function invocations	sin(theta)
unary expressions	-x
binary expressions	4.5+8.0, 5<=6

Binary operator precedence (decreasing order)

^
AND * / |
OR + -
&
> < = >= <= !=

Function commands

Form <*var*> = <*command:qualifiers(parameters)*>
e.g. w = read:ascii:columns("data.dat", "x,y,e")
A command invoked as a function must always specify a parameter list, even if it is empty e.g. x=drawit().

Multi line commands

Case sensitivity

All Genie keywords must be in uppercase and should not be abbreviated. Procedure and variable names are case insensitive.

IF statement

```
IF <condition1>
        <statement 1>
[ELSEIF <condition2>
        <statement2>]
[ELSE
        <statement3>]
ENDIF
```

CASE statement

```
CASE <expression>
        IS <value1> [TO <range1>]
                <statement1>
        IS <valueN> [TO <rangeN>]
                <statementN>
        OTHERWISE
                <statementN+1>
ENDCASE
```

When <expression> evaluates to any of the specified values (or range of values) the associated statement is executed. If there is no match and no OTHERWISE statement, no action is taken.

LOOP statement

```
LOOP [<var> FROM <start> TO <end> [STEP <siz>]]
        <statements>
        EXITIF <condition>
        <statements>
ENDLOOP
```

The loop counter variable does not need to be declared anywhere else.

Procedures

These are normally loaded from a file which the load command e.g. load "myprocs.gcl". Once loaded, the procedure names may be invoked as either keyword or function commands.

Statement order

```
PROCEDURE <name>
# The QPR group must come before the GL group
[QUALIFIERS/<qname1>...../<qnameN>]
[PARAMETERS <pname1=ptype1>....]
[RESULT <rname=default value>]
# The GL group must come before the statements
# that comprise the body of the procedure
[GLOBAL <gvar1> [ = <initial value>] .....]
[LOCAL <lvar> [ = <initial value>] ......]
<statements>
ENDPROCEDURE
```

Qualifiers

May be tested for in the procedure by treating them as variables with a truth value e.g. if '/Markers' was a qualifier, 'IF Markers' would test for its presence.

Parameters

Simple variables are passed by value. Arrays and workspaces (and hence their contents) are passed by reference. Specifying the parameter type is optional.

Result

A variable returned by the procedure when it is called as a function

Global variables

Shared with other procedures which declare global variables of the same name. Initialised by first procedure which supplies an initial value.

Local variables

Exist only within a procedure during its lifetime.

Data types

Constants
Must begin with the '$' character.

Variables
There is no implicit typing of variables and variables can change type at any time. Reals are always double precision floating point numbers, specified with a decimal point or in scientific notation. Literal strings must be enclosed in double quotes.
The operators ^ (power), +,-,*,/,<,>,<=,>=,!= (not equal to) and the functions sin, cos, tan, arcsin, arccos, arctan, log, ln, exp, alog are applicable to both real and integer arguments. The | (remainder) operation is provided for use with integers. String operators are + (concatenate) and <,>,=,<=,>=,!=. The latter operate on string length except when the strings are of equal length, when collation order is used.

Special string characters

\"	double quote	\\	single backslash
\n	newline	\t	tab
\f	formfeed	\v	vertical tab
\r	carriage return	\0xx	octal byte value
\Xnn	hexadecial byte	\nnn	decimal byte value

Arrays
Created with the dimensions function
<arrayname> = dimensions(dim1,dim2,...dimn)
The array type is determined by the first assignment to an element of the array and cannot then be changed.
e.g. myarray[3,3] = 2.3 # array type fixed as real now !
Note the use of square brackets and integer indices. All indices start at 1. Valid array types are real, integer, string and workspace. Storage order is row major (i.e. the opposite of Fortran !)

Array operators
Entire arrays can be operated on without resort to looping through individual element (*see next page...)*

Array operators (continued)
Unary operators are applied individually to each element of an array. Binary operators are applied to corresponding elements of two arrays, the correspondance being determined by storage order when the arrays are of different dimensionality. Resultant array size is determined by the array on the lhs.

+ - (unary)	e.g. y = -x
+ - * / | ^ (binary)	e.g. z = x * y
< ,>,<=,>=	shorter than, longer than etc.
=	elements and lengths identical
!=	elements and / or lengths differ
&	append rhs array to lhs array

All trigonometric and transcendental functions are applicable and function as unary operators e.g. y=sin(x). The length() function returns the total length of an array.

Workspaces
Analagous to a 'C' structure, a workspace may contain many individual elements of different types. They are created with the fields() function e.g. w1=fields(). Individual fields are then created by assignment e.g.

```
w1=fields()
w1.title="My workspace"  # string variable
w1.xvals=dimensions(4)   # create an array
w1.xvals[1]=10           # array is integer type
```

Workspace operators
A set of operators analogous to the array operators listed above is available for Genie2 type workspaces but obviously these will operate on specific *fields* within the workspaces, and so will be listed on a separate reference card. They are documented in the OpenGenie manual.

OpenGenie v1.0

Genie command language quick reference card

The Genie Tutorials

March 1996