ELIST$ - List Directory (Extended)

The ELIST$ routine is available to obtains details of the first or next file present in a GSM directory previously opened using the OPEN$ or OPENS$ routine. The ELIST$ routine returns extra information that is not available with LIST$.

1. Invocation

To obtain details of the first or next file present in a directory code:

```
CALL ELIST$ USING filename type
```

where filename is the name of an FD opened using OPEN$, or OPENS$; and type is the name of a PIC S9(2) COMP field in which a value defining the file type, as described below, is returned when ELIST$ returns normally or with an exception 3 or 4.

2. STOP Codes and Exception Conditions

The following STOP codes may be generated by ELIST$:

STOP code	Description
12202	The FD has not been opened by OPEN$ (or OPENS$).

The following EXIT codes may be returned by ELIST$:

EXIT code	$$COND	Description
12201	1	An irrecoverable I/O error has occurred.
12202	2	No more files are present in the directory.
12203	3	The current file is OPEN shared.
12204	4	The current file is OPEN exclusively.

3. Programming Notes

The family of routines OPEN$, OPENS$, LIST$, ELIST$ and CLOSE$ can be used to determine the file-id and type of each file present on a Global System Manager direct access volume. OPENS$, rather than OPEN$, must be used if the unit is in use as a spool unit.

The OPEN$, OPENS$ LIST$, ELIST$ and CLOSE$ routines all require a filename as their first parameter. This name identifies an FD to be used in the directory processing operation. At a minimum you must code the FD statement and the following ASSIGN statement:

```
FD filename ORGANISATION organisation
ASSIGN TO UNIT unit-id FILE file-id [VOLUME volume-id]
```

You can use any convenient organisation (e.g. UNDEFINED, RELATIVE-SEQUENTIAL) since the one specified is immaterial as far as the directory routine is concerned. You should specify the file-id as a symbol, since ELIST$ returns each file-id found to be present in this field. The volume-id should be specified as a symbol if you wish to examine it following a call of OPEN$.

The FD must be closed when it is passed to OPEN$. It will then be opened so that it can be processed by ELIST$ or CLOSE$: the type of open is special, however, and prevents the FD from being used by any other file processing operation such as a READ or WRITE. When you have finished examining the directory you must close the FD using CLOSE$. It is then returned to the normal closed state and can, should you so wish, be processed by a conventional access method OPEN statement.

If no exception is returned the volume-id of the currently mounted volume will have been placed in the field you have specified using the FD. Note that this means that if the volume-id is returned and the file subsequently opened normally, then volume-id checking will take place unless you zeroise the volume-id field.

The possible values of the type field returned when the routine signals normal completion or exception 3 or 4 are listed in the flowing table. In this case the file-id is returned as well, in the field named in the FILE clause of the FD's ASSIGN statement.

Type	**Mnemonic**	**Description**
-1 to -99		Backup file. If the type is -1 this is the first file of the cycle, for -2 the second file, and so on.
0	RS	Relative sequential file
1	IS	Indexed sequential file
2	PL	Program file or library
3	TF	Text file
4	VL	Variable length record file (e.g. AutoClerk control file).
5	DL	Data library
6	DM or CL	Compilation file or library (if file has C. prefix), or DMAM database file (otherwise).
7	BP	Global Planner plan file
8	WP	Global Writer document file
9	DB	Global Finder database
11 to 99		User-dependent organisation. Files of type 11 to 99 can be created using the basic direct access method with the appropriate type specified in the ORGANISATION statement used to produce the access method.
80	ST	Speedbase TAP file
81	81	Reserved for future use
82	82	Reserved for future use
83	83	Reserved for future use
84	84	Reserved for future use
85	OR	Reserved for future use
86	86	Reserved for future use
87	87	Reserved for future use
88	DC	$BACKUP extension file
89	D2	Temporary Speedbase Dictionary file
90	SF	Speedbase full backup file
91	SI (sic)	Speedbase full incremental backup file
92	SJ	Speedbase part incremental backup file
93	93	Reserved for future use

94	94	Reserved for future use
95	SX	Extended Speedbase Unix C-ISAM database schema file
96	SN	Speedbase Btrieve database schema file
97	SU	Speedbase Unix C-ISAM database schema file
98	DD	Speedbase data dictionary
99	SB	Speedbase database
100	SA	Save file created by the $F SAV instruction
101	BO	Physical bootstrap
102	BE	Bootstrap file or library
103	MN	Monitor, Nucleus file or library
104	IN	Configuration file
105	SW	Swap file
106	CW	Compiler work file
107	US	User file
108	SP	$SPOOL schedule file
109	CF	Global 2000 software integrity file
110	LG	System Log file
111	SK	Data skeleton
112	SY	System file
113	PF	Partially created file
114	LB	Data file library
115	IF	Integrator file
116	SC	Schema file (TFSCM$)
117	SI (sic)	Schema file (SCHEM$)
118		Reserved for future use
119		Reserved for future use
120		Reserved for future use
121		Reserved for future use
122		Reserved for future use
123		Reserved for future use
124		Reserved for future use
125		Reserved for future use
126		Reserved for future use
127		Reserved for future use

ELIST$ also returns additional information in the following redefinition of the FD:

```
01    FILLER REDEFINES filename
  02  FILLER      PIC X(48)
  02  FDCRE       PIC 9(6) COMP          * FILE CREATION DATE
  02  FILLER      PIC X(17)
  02  FDSTA       PIC 9(9) COMP          * FILE START ADDRESS
  02  FDFIL       PIC 9(9) COMP          * FILE SIZE
  02  FDEXT       PIC 9(9) COMP          * FILE EXTENT
```

Exception condition 1 will be returned if an irrecoverable I/O error occurs, and exception condition 2 when there are no more files present in the directory. In the case of either exception the file-id and type fields are not updated. You should follow either of these exceptions by issuing a call on the CLOSE$ function to re-establish the FD's initial status and release the Global System Manager resources involved in the directory processing operation.

Each call on the ELIST$ function which completes normally, or with exception 3 or 4, returns information concerning a single file from the directory. If the file is not in use (i.e. not already open by the current job or a competing job in a multi-user environment) the routine signals normal completion. Exception 3 is returned if the file is open as shared, and exception 4 if the file is open exclusively.

The file information is supplied in the same order in which it would appear were the directory to be listed using the file utility's LIS instruction: file-ids do not appear in alphabetical order, nor does the sequence of presentation necessarily reflect the time at which a file was created or its position on the volume. Once exception condition 2 is returned all file information has been returned to you. If you require to scan the directory again you should follow the CLOSE$ call with a new OPEN$ call and a new sequence of ELIST$ calls.

Directory operations are relatively slow, so whenever possible you should use conventional access method open operations to determine whether or not files are present. For example, to check whether a file of known type is present it is usually best to issue an OPEN OLD or OPEN SHARED for it.

The directory operations are best employed in applications which are not performance critical, such as displaying or listing file information in response to an operator enquiry, or in printing or conversion operations where a number of files on the same volume are subjected to lengthy processing. In both these cases the time spent accessing the directory is short in comparison with the display or file processing time.

4. Examples

[EXAMPLE REQUIRED]

5. Copy-Books

None.

6. See Also

OPEN$	Open Directory
OPENS$	Open Spool Directory
LIST$	List Directory
CLOSE$	Close Directory