SUPERVISOR and LOG contexts HELP

Relative to SUPERVISOR version 42.420.75

MAGUS version 42.420.12

This HELP file is intended to provide Metalogic SUPERVISOR users with current and useful information on the features and capabilities of the new LOG context implementation. Please note that the METANOTES data files for SUPERVISOR, OPAL and OPAL/GSTABLEGEN, available on your usual Metalogic release tape, contain useful and late-breaking information which may not yet be included in this HELP document.

Introduction

The LOG contexts implementation allows the real-time capture of various types of log records by Supervisor, as they are written to the SUMLOG file by the MCP. Supervisor has various new OPAL program contexts which specifically allow access to such log record types and in particular, OPAL SITUations can be used to provide powerful and flexible filtering facilities. Associated with these filters, OPAL ODTSequences can be used to automatic operator responses and/or generate tailored reports.

Alternatively, using the EVAL command, Supervisor can emulate LOGANALYZER to scan single or multiple SUMLOGs, again using OPAL SITUations for filtering purposes and ODTSequences for reporting or taking operator action. This feature is very flexible and Supervisor's log reading capabilities are significantly faster than most other utilities.

The LOG contexts currently available allow retrieval of information from the following log record types. Each of these contexts has its own unique identifier which is used in the program definition. SUPERVISOR uses an Unisys provided library entrypoint in the MCP, called REPORT_LOG_ENTRIES, to provide these facilities.

File open and close (FILEOPEN, FILECLOSE), programmatic database open and close (DATABASE), security violation entries (SECURITY), MCS security violation entries (MCSSECURITY), print request maintenance (LOGPR), log identity records (EI), station log-on and log-off (LOGON, LOGOFF), end-of-job and end-of-task log entries (LOGEOJ) and beginning-of-job and beginning-of-task log entries (LOGBOJ). Lastly a generic program type called LOG, allows access to ALL SUMLOG record types. Some of these contexts are allowed optional subcontext assignments which provide powerful filtering to restrict the types of records returned to Supervisor by the MCP. The full list of currently available Log contexts are shown below:

Contexts	Optional subcontexts
LOG	**Major type and Minor type list**
FILEOPEN	**None**
FILECLOSE	**CLOSE, PLI**
LOGPS	**STARTED, COMPLETION,PRINTED, FINISHED**
SECURITY	**None**
DATABASE	**OPEN, CLOSE**
LOGBOJ	**BOJ, BOT**
LOGEOJ	**EOJ, EOT**
LOGON	**None**
LOGOFF	**None**
MCSSECURITY	**None**
MSG	**None**

Any Major type with multiple Minor Although not all SUMLOG record types are covered by a Supervisor LOG types context equivalent, it is Metalogic's intention that more types will be added in the future according to demand. However, Supervisor does have the CLOSE, PLIcapability to examine ALL records types using the basic LOG context type. COMPLETION, CREATED
In this case, additional filtering controls can be provided in the SITUation PRINTED, STARTEDdefinition to help minimise the overhead of tracking every record type.

Please note that Supervisor's capability to capture LOG records is independent of the systems LOGGING options. It does not matter if the LOGGING command has disabled the recording of BOJ and BOT records to the SUMLOG; Supervisor will still receive log event information from the MCP for any active **WHENs** using a LOGBOJ context

The previous statement does not apply to the **EVAL** command. If the LOGGING command has disabled both BOJ and BOT records, then an **EVAL** of a LOGBOJ SITUation will not find any entries. This is because Supervisor must read one or more SUMLOGs to search for BOJ information but if the log records are not present then there is little that can be achieved.

Please note that, unlike the Unisys LOGANALYZER product, Supervisor **EVAL**s that read multiple Sumlog files will return any entries found in reverse chronological order.

Further, since the MESSAGE context is now a LOG context, has been included for completeness because when used with the EVAL command, Supervisor will scan the required SUMLOGs. Previously, the MSG context was only available for the event-based WHENs and not EVALs.

Each of the above contexts have their own attribute subsets making it practical to write simple SITUations to track, filter, report and action many varieties of system activities. Where possible, OPAL attributes map directly onto fields in the relevant log entry with similar and meaningful names. For more details on the attributes associated within each context, the **TT HELP ATT =:<context>** command from an ODT or Supervisor window can be used in conjunction with provided references to the Unisys System Log Programming Reference manual.

For example, to see all attributes associated with the SECURITY context, try **TT HELP ATTR =:SECURITY**. To see information about individual attributes, use **TT HELP ATTR <attribute>**.

All of these new contexts can be used in real-time (as WHENs) or to scan the SUMLOG (using the **EVAL** command). Please see the sections concerning changes to the Supervisor **EVAL** and **WHEN** commands for more information.

SECURITY context

The SECURITY context corresponds to log entries detected by the LOG SECURITY command whose type are Major type 6 (Miscellaneous), Minor type 4 (Security Violation entry). Using the appropriate attributes, OPAL programs are capable of handling system security violations, as they occur, and taking action or reporting transgressions immediately.

A typical SITUation to capture an unauthorised attempt to open a PRIVATE file:

```
DEFINE + SITUATION PRIVATE_FILE(SECURITY):
     VIOLATIONCODE EQL PRIVATEFILE

DEFINE + ODTSEQUENCE PRIVATE_FILE(SECURITY):
     RECORD[5]("User ",USERCODE,,"(",MIXNUMBER,,") ",,
               " attempted open of PRIVATE file");
     RECORD[5]("Filename is: ",,VIOLATIONNAME);

WHEN PRIVATE_FILE DO PRIVATE_FILE
```

Example output:

```
STIRLING1 11:21:00: User META  (4567) attempted open of private file
STIRLING1 11:21:00: Filename is  (FLEX)VERY/PRIVATE/FILE ON PACK
```

Each line of output is prefixed by the originating hostname and time-of-day.

In the above example, the SITUation is only looking for a specific type of security violation using the VIOLATIONCODE attribute. In this case, the target is any PRIVATEFILE violation, which is an internal OPAL mnemonic value for this security code. The other codes can be seen using the command

```
TT HELP ATTR VIOLATIONCODE
```

Note that MIXNUMBER is not actually a SECURITY context attribute; it is a synonym for LOGMIXO which actually belongs to the universal LOG context. All attributes that are members of the LOG context may be used in any of the other log contexts regardless of scope.

FILEOPEN and FILECLOSE contexts

The FILEOPEN and FILECLOSE contexts allow retrieval of programmatic file open (Major type 1 and Minor type 3) and file close (Major type 1 and Minor type 4) log records. Sometimes, it is useful to know when a new file has just been created on the system, e,g. for file transfer purposes,

There have always been some limitations in file log records; in particular, the USERCODE and ACCESSCODE of the task opening or closing the file is not logged. If using the EVAL command, then Supervisor will also have this limitation but this is not the case for OPAL programs driven by the WHEN command. Using the OPAL VIA intrinsic, any information about the acting job or task can be easily retrieved form the current mix using the LOGMIXNO attribute, as if the program was actually a MX context. An example of this capability is shown below. Remember that you can only do this in a real-time WHEN environment, whilst the job or task is still active.

```
DEFINE + SITUATION LOG_OPEN(FILEOPEN):
   KIND NEQ PORT AND INTNAME="TAPELOG"

DEFINE + ODTSEQUENCE LOG_OPEN(FILEOPEN):
   RECORD[9]("OPEN",,LOGJOBNO,"/", LOGMIXNO,, VIA(LOGMIXNO:#(NAME
           ,,"(",USER,")")));
   RECORD[9](" " 9,INTNAME,"=",TITLE
          ,IF FAMILYNAME NEQ "" THEN #(" ON ",FAMILYNAME) ELSE "");
   RECORD[9](" " 9,"FILE SIZE ",
          FILESIZE,,"SECTORS");

WHEN LOG_OPEN DO LOG_OPEN
```

Example output:

```
STIRLING1 14:22:06 OPEN 5678/5679 *SYSTEM/TESTER 1457
STIRLING1 14:22:06          INTNAME=TESTFILE ON DISK
STIRLING1 14:22:06          FILE SIZE 3024 SECTORS
```

A similar example of shown below for detecting a PORT file open. Note that many of the attributes for a port file open are not applicable for all open records. For example, the YOURUSERCODE and YOURHOSTNAME port-only attributes will always return null strings if used to provide information about a disk file open. In the example below, a DISPlay program is being to report formatted information back to the originating station or to printer.

```
DEFINE + SITUATION LOG_PORTOPEN(FILEOPEN):
   KIND =PORT

DEFINE + DISPLAY LOG_PORTOPEN(FILEOPEN):
  TIME(LOGTIME),,"OPEN",,LOGJOBNO,"/", LOGMIXNO,,VIA(LOGMIXNO:NAME)
  " " 9,"EXT NAME ",TITLE,/,
  " " 9,"INT NAME",,INTNAME,/,
  " " 9,"FAMILY NAME:",FAMILYNAME,/,
  " " 9,"FILE ACCESS RULE =",ACCESSRULE,,
  IF ACTOR_STACKNO=DECLARER_STACKNO THEN
    #("(ACTOR = DECLARER)",/,
      " " 9,"STACK ",HEXSTRING(ACTOR_STACKNO),,"JOB ",ACTOR_JOBNO,,
      "TASK ",ACTOR_TASKNO)
  ELSE
    #(/,
       " " 9,"(ACTOR = STACK ",HEXSTRING (ACTOR_STACKNO),,"JOB "
       ,ACTOR_JOBNO,,"TASK ",ACTOR_TASKNO,")",/,
       " " 9,"DECLARER = STACK ",HEXSTRING (DECLARER_STACKNO),,
       "JOB ",DECLARER_JOBNO,,"TASK ", ACTOR_TASKNO),/,
  " " 9,"YOURHOSTNAME: ",YOURHOSTNAME,,"MYPABN0 = ",MYPORTADDRESS,/,
  " " 9,"YOURUSERCODE: ",YOURUSERCODE,,"MYSABN0 = ",MYSUBPORTADDRESS,/
  " " 9,"YOURNAME    : ",YOURNAME,,   "SUBFILE INDEX ",SUBFILEINX,/,
  " " 9,"SERVICE     : ",SERVICE,,"PROVIDERGROUP:  ",PROVIDERGROUP,/,
  " " 9,"PROVIDER SELECTED: ",PROVIDER,/,
  " " 9,"TIMELIMIT     : ",CONNECTTIMELIMIT,,"INIT PRIMATIVE ",
                     DROP(#(INITPRIMATIVE),5),/,
  " " 9,"RESPONSE TYPE : ",RESPOND,,"USAGE ",MYUSE
```

Example output might appear as follows on a Supervisor window or remotespo:

```
15:53:48 OPEN 877/1106 *METALOGIC/SUPERVISOR/HOTLINE ON REST
         EXT NAME RECORDERHOTLINE
         INT NAME HOTLINE
         FAMILY NAME:
         FILE ACCESS RULE =DECLARER (ACTOR = DECLARER)
         STACK 2E6 JOB 877 TASK 1106
         YOURHOSTNAME: STIRLING2 MYPABN0 = 131
         YOURUSERCODE:  MYSABN0 = 126
         YOURNAME    : RECORDER SUBFILE INDEX 1
         SERVICE:    : BNANATIVESERVICE
         PROVIDERGROUP :
         PROVIDER SELECTED: BNAV
         TIMELIMIT : 0 INIT PRIMATIVE OPEN
         RESPONSE TYPE : NOTINVOKED   USAGE UNKNOWN (3)
```

The FILECLOSE context is indeed similar to FILEOPEN but the subset of attributes are slightly different since the format of these log records are not identical. For example, the CLOSETYPE attribute is only valid for FILECLOSE and OPENERROR is only applicable for FILEOPEN.

FILECLOSE also has two subcontexts called PLI and CLOSE and a SITUation definition without a subcontext means that Supervisor will capture event information about both. The PLI subcontext is actually a Major Type 1, Minor Type 10 log entry and contains usage information for open files if interval logging has been requested by using the PLI (Periodic Logging Interval) system command. The FILE option in LOGANALYZER can be used to obtain a list of these entries, if present, as well as the usual open and close file entries. The format and contents of this log entry are almost identical to those of the Major Type 1, Minor Type 6 (File Close) with some minor differences.

The PLI subcontext is useful for tracking file statistical information over time for applications that keep files open over long periods of time.

In the following LOG_CLOSE programs, RECORD[6] is used to send information out to an alert program via the RECORDER/HOTLINE interface.

```
DEFINE + SITUATION LOG_CLOSE(FILECLOSE=CLOSE):
   TRUE

DEFINE + ODTSEQUENCE LOG_CLOSE(FILECLOSE):
   RECORD[6]("CLOSE",,LOGJOBNO,"/", LOGMIXNO,,VIA(LOGMIXNO:#(NAME
            ,,"(",USER,")")));
   RECORD[6](" " 9,INTNAME,"=",TITLE
            ,IF FAMILYNAME NEQ "" THEN #(" ON ",FAMILYNAME) ELSE ""
   RECORD[6](" " 9,"PHYSICAL(READS=",READS,,"WRITES=",WRITES,")");
   RECORD[6](" " 9,"PERM AREASIZE ",AREASIZE," SECTORS, FILE SIZE "
             FILESIZE,,"SECTORS");

WHEN LOG_CLOSE DO LOG_CLOSE
```

In the LOG_CLOSE ODTSequence, the LOGMIXNO attribute has been used inside the VIA function allowing the retrieval of the task usercode and name. CLOSE records do not provide this information but VIA can be used to dynamically "fetch" information from the MX context using the mixnumber of the task responsible for the creation of this record.

LOGON and LOGOFF contexts

The LOGON (Major type 4, Minor type 1) and LOGOFF (major type 4, Minor type 2) contexts allow the tracking of MCS session log-on and log-off entries. Typically, all MCSes, such as COMS and CANDE, create these records whenever an individual user initiates or terminates a "session". Indeed, Supervisor will create its own log-on and log-off records for WHENs, DOs and AFTERs subject to the settings of his MONITORING and LOGMINIMAL internal options.

The following example shows some OPALs that capture all session log-off session records and, in this case, send messages to an external alert program, via the RECORD [8] construct.

```
DEFINE + SITUATION LOG_LOGOFF(LOGOFF):
   TRUE

DEFINE + ODTSEQUENCE LOG_LOGOFF(LOGOFF):
   RECORD[8]("LOGOFF",,LOGMIXNO,,TERMINATION,,"ORIGINATING LSN ",LSN);
   RECORD[8](" " 9,"USERCODE ",USERCODE);
   IF CHARGECODE NEQ EMPTY THEN
      RECORD[8](" " 9,"CHARGECODE ",CHARGECODE);
   IF ACCESSCODE NEQ EMPTY THEN
      RECORD[8](" "9,"ACCESSCODE ",ACCESSCODE);
   RECORD[8](" " 9,"ELAPSED   TIME: ",TIME(ELAPSED));
```

Similarly, the following DEFINEs can be used to capture MCS log-on records; in this case a RECORD [7] statement is used to send information out to a RECORDER/HOTLINE alert program. This time, however, we are only interested in a log-on records for the usercode META and accesscode of TEST using a simple condition in the SITUation.

```
DEFINE + SITUATION LOG_LOGON(LOGON):
   USERCODE = "META" AND ACCESSCODE ="TEST"

DEFINE + ODTSEQUENCE LOG_LOGON(LOGON):
   RECORD[7]("LOGON",,LOGMIXNO,,CLASS,,"ORIGINATING LSN ",LSN);
   RECORD[7](" " 9,"USERCODE ",USERCODE," ACCESS ",ACCESSCODE);
   IF CHARGECODE NEQ EMPTY THEN
      RECORD[7](" " 9,"CHARGECODE ",CHARGECODE);
```

This sort of filtering is very powerful, allowing the construction of simple rules to pinpoint only those log records that need to be recorded.

MCSSECURITY context

The MCSSECURITY context (Major type 4, Minor type 6) is similar to the SECURITY context except that the provided security violations are generated by the controlling MCS rather than the MCP. These security violations are usually applicable for individual users and will also document what action was taken by the MCS after the violation has occurered e.g. the station was cleared, saved etc.

As with earlier examples, the following programs can be used to record all MCS security violations, in real-time, if desired, to the RECORDER/HOTLINE programs. In this case, RECORD subfile 11 has been used.

```
DEFINE + SITUATION LOG_MCSSEC(MCSSECURITY):
   TRUE

DEFINE + ODTSEQUENCE LOG_MCSSEC(MCSSECURITY):
   RECORD[11]("MCS SECURITY VIOLATION",,LOGMIXNO);
   RECORD[11](" " 9,MCS_ERRORTEXT,," CODE: ",MCS_ERRORCODE);
   RECORD[11](" " 9,"ORIGINATING LSN:",LSN,,STATIONNAME);
   RECORD[11](" " 9,"USERCODE       :",USERCODE);
   RECORD[11](" " 9,"ERROR ITEM     :",INPUT);
```

The MCS_ERRORTEXT attribute is unusual for an OPAL attribute in that it will return a textual description of the encountered security violation. For example:

```
STIRLING1 12:25:48 MCS SECURITY VIOLATION 4098
STIRLING1 12:25:48 Invalid accesscode/password at log-on  CODE: 5
STIRLING1 12:25:48 ORIGINATING LSN: 45  ET21/CANDE/1
STIRLING1 12:25:48 USERCODE: META
STIRLING1 12:25:48 ERROR ITEM: META
```

DATABASE context

The DATABASE context refers to all application database open and close log records (Major type 1, Minor types 19 and 20 for open and close respectively). Note that this only applies at the application level, i.e. each time a program performs a programmatic database open or close then the MCP will log these records. If you wish to track database start and termination, then its BOJ and EOJ should be tracked by the LOGBOJ and LOGEOJ contexts respectively.

There are two valid subcontexts for the DATABASE context, called OPEN and CLOSE, controlling the filtering of database open or close log records. If no subcontext is provided, then both record types will be tracked and attributes are valid for both types. The integer attribute OPENTYPE can be used to differentiate between them (OPENTYPE=DBOPEN for OPEN records and OPENTYPE=DBCLOSE for CLOSE records).

```
TT DEFINE + SITUATION DBOPEN_ONLY(DATABASE=OPEN):
```

```
TT DEFINE + SITUATION DBCLOSE_ONLY(DATABASE=CLOSE):
```

The information accessible via a DATABASE context is essentially that returned by a LOGANALYZER run, using the DATABASE modifier, and has useful information that can be extracted. The name of the database, the name of the invoking program and its usercode or accesscode are particularly interesting in tracking database access.

```
DEFINE + SITUATION LOG_DATABASE(DATABASE=OPEN):
   DBNAME INCL "METATAPELIB"
```

```
DEFINE + ODTSEQUENCE LOG_DATABASE(DATABASE):
   RECORD[7]("OPEN OF ",DBNAME,"(",DBMIXNO,")" DETECTED")
   RECORD[7]("TASK NO:",LOGMIXNO,,"NAME : ",NAME);
   RECORD[7]("PROG INTNAME : ",INTNAME);
   RECORD[7]("USER: ",USERCODE,," ACCESSCODE: ",ACCESSCODE);
```

The SITUation filters out all database activity other than any those whose name includes the partial string, "METATAPELIB" and only database open records.

Example output:

```
STIRLING1 08:32:17 OPEN OF METATAPELIB (3029) DETECTED
STIRLING1 08:32:17 TASK NO: 4133 NAME: METALOGIC/OPALTAPELIB
STIRLING1 08:32:17 PROG INTNAME : TAPEDB
STIRLING1 08:32:17 USER: TAPELIB  ACCESSCODE: LIBRARIAN
```

LOGEOJ context

Many Supervisor users will be well aware that a COMPLETED context already exists for trapping EOJ and EOT events, so why have a log-based alternative? The primary reason for this is that the set of attributes for a COMPLETED context is significantly less than its log-based counterpart. For example, normal EOJ resource statistics such as CPU time, I/O time, core usage etc. cannot be retrieved by a COMPLETED OPAL program.

If the context name, LOGEOJ, is used on its own, both EOJ and EOT records will be tracked by Supervisor. However, there are two subcontexts which may be used to eliminate either EOJ or EOT if desired. For example:

```
TT DEFINE + SITUATION EOT_ONLY(LOGEOJ=EOT):

TT DEFINE + SITUATION EOJ_ONLY(LOGEOJ=EOJ):
```

The TT HELP ATT =:LOGEOJ and TT HELP ATT =:COMPLETED commands can be used to see the differences between the two attribute subsets. The LOGEOJ context supports all of the COMPLETED context attributes except for the following:

JOBMESSAGES, OPTION, OPTIONS and STOPPOINT subset

The above attributes are considered to be of trivial importance and will not be supported in the newer LOGEOJ implementation.

In the following example, the

```
DEFINE + SITUATION LOGEOJ(LOGEOJ):
   USER = "META" AND ACCESSCODE ="TEST"

DEFINE + ODTSEQUENCE LOGEOJ(LOGEOJ):
  RECORD[8]("Jobno     : ",JOBNUMBER 17,,   "Taskno    : ",MIXNUMBE
  RECORD[8]("Name      : ",NAME 40);
  RECORD[8]("Usercode  : ",USERCODE 17,,
            "Accesscode: ",ACCESSCODE 17);
  RECORD[8]("Cards read: ",CARDSREAD 17,,"Lines     : ",LINESPRINTE
  RECORD[8]("Elapsed   : ",TIME(ELAPSEDTIME) 17,,
            "EOJ type  : ",EOJTYPE);
  RECORD[8]("Init pbit : ",INITIALPBITS 17,,
            "I/O time  : ",TIME(IOTIME));
  RECORD[8]("Queue    : ",QUEUE 17,,       "SourceMCS :",SOURCEMCS
  RECORD[8]("Starttime: ",TIME(STARTTIME) 17,,
            "Startdate :",DATETOTEXT(STARTTIMEDAY,DDMMYYYY));
  RECORD[8]("Max ASDs : ",MAXASDS 17,,      "Chargecode: ",CHARGECO
```

A typical example of the output from the above RECORD statements might appear on a HOTLINE station as follows:

```
STIRLING1 09:27:11 Jobno     : 1234          Taskno    : 1235
STIRLING1 09:27:11 Name      : *SYSTEM/DCALGOL
STIRLING1 09:27:11 Usercode  : META          Accesscode: TEST
STIRLING1 09:27:11 Cards read: 280           Lines     : 3026
STIRLING1 09:27:11 Elapsed   : 00:28:14      EOJ type  : NORMALEOTV
STIRLING1 09:27:11 Init pbit : 549           I/O time  : 00:00:44
STIRLING1 09:27:11 Queue     : 9             SourceMCS : 4
STIRLING1 09:27:11 Starttime : 14:45:55      Startdate : 05/09/1996
STIRLING1 09:27:11 Max ASDs  : 210           Chargecode: MYCHARGE
```

Note the use of the TIME and DATETOTEXT functions to convert raw time and date values into a more meaningful format. The EOJTYPE attribute returns a mnemonic value indicating information about the tasks termination i.e. whether it was terminated normally or abnormally by operator, system or programmatic reasons. In the above case, the value is NORMALEOTV which indicates, unsurprisingly, that this was a normal EOJ. Abnormal terminationswill usually return DSEDV or STEDV values and, in these cases, the EOJCAUSE and EOJREASON attributes can be used to provide additional information about the exact nature of the DSed task.

LOGBOJ context

Similar to its sibling, LOGEOJ, the LOGBOJ context is now the preferred context used to track beginning-of-task (BOT) and beginning-of-job (BOJ) events on the system. As with LOGEOJ, the LOGEOJ context has several distinct advantages over the old SITUation type of MX=BOT or MX=BOJ. With a MX=BOJ SITUation, Supervisor handles BOT/BOJ events by requesting mix information from the MCP using a standard GETSTATUS interface. This MCP information call has a small overhead and is totally reliant on the task being active at the time of the request. If the task is NOT active, usually because it has already terminated, Supervisor will not be able to correctly set up the appropriate MX attributes or the event may even be discarded.

Indeed, if the system is very busy with lots of event activity and Supervisor is running a large number of WHENs, it is possible for BOJ notices to be lost even though the task may have a reasonable elapsed time.

The LOGBOJ context totally avoids these problems; because the log record already has the information it needs, the MCP information call is not required and the user task need not be active at the time Supervisor is processing the information.

However, one disadvantage does exist with the LOGBOJ context; the attribute subset compared to the MX subset is much smaller. This is because the information for a LOG BOJ record is very specific and there is no provision for needing to know information about ASDSINUSE or ACCUMIOTIME. Since the BOJ is only recording the instant that the job starts, this sort of information would likely be zero.

```
DEFINE + SITUATION LOGBOJ(LOGBOJ):
   USER = "META" AND ACCESSCODE ="TEST"

DEFINE + ODTSEQUENCE LOGBOJ(LOGBOJ):
  RECORD[8]("Jobno      : ",JOBNUMBER 17,,   "Taskno     : ",MIXNUMBE
  RECORD[8]("Usercode   : ",USERCODE 17,,
            "Accesscode: ",ACCESSCODE 17);
  RECORD[8]("Name       : ",NAME 40);
  RECORD[8]("Queue      : ",QUEUE 17,,       "Sourcename: ",SOURCENA
  RECORD[8]("Chargecode: ",CHARGECODE);
  RECORD[8]("Family     : ",VIA(TASK(MIXNO):FAMILY));
  RECORD[8]("Identity   : ",VIA(TASK(MIXNO):IDENTITY));
  IF PPED THEN
     RECORD[8]("Task is a PRIVILEGED program");
  IF CPED THEN
     RECORD[8]("Task is a CONTROL program");
```

As with previous examples, the output might appear as on a HOTLINE station as follows:

```
STIRLING1 19:26:34 Jobno     : 1234            Taskno    : 1235
STIRLING1 19:26:34 Name      : *SYSTEM/DUMPALL
STIRLING1 19:26:34 Usercode  : META            Accesscode: TEST
STIRLING1 19:26:34 Queue     : 9               Sourcename: ODT/CANDE/
STIRLING1 19:26:34 Chargecode: MYCHARGE
STIRLING1 19:26:34 Family    : PACK OTHERWISE DISK
STIRLING1 19:26:34 Identity  : DUMPALL
STIRLING1 19:26:34 Task is a PRIVILEGED program
STIRLING1 19:26:34 Task is a CONTROL program
```

Note that in the above example, where inaccessible attributes might possibly be needed (e.g. the MX attributes **FAMILY** and **IDENTITY)**, the VIA function can be used to extract this information from the MX context using the reference attribute **TASK**. Reference attributes such as **TASK**, **UNIT** and **TAPEDB** allow access to information in the MX, PER and TAPEDB contexts, respectively, from any other OPAL program context.

LOGPS context

The LOGPS context allows retrieval of various system PrintS request event information corresponding to Sumlog record Major type 1 and Minor types 11 (Print Request Created), 12 (Print Request Complete), 13 (Start Printing File) and 14 (Finish Printing file). This context allows users to track the creation and printing of PrintS requests in real-time allowing automatic printing and modification of existing requests. These Minor types can be individually selected using the following subcontexts:

```
TT DEFINE + SITUATION PS(LOGPS=CREATED):        Minor type 11
TT DEFINE + SITUATION PS(LOGPS=COMPLETION):     Minor type 12
TT DEFINE + SITUATION PS(LOGPS=STARTED):        Minor type 13
TT DEFINE + SITUATION PS(LOGPS=PRINTED):        Minor type 14
```

The CREATED subcontext refers to the initial creation of the print request (i.e. the addition of the first print file into a new PrintS request). The STARTED and PRINTED subcontexts indicate the progress of the physical printing of the specified request. The COMPLETION phase refers to the release of the request to the PrintS system; this is usually due to PRINTSDISPOSITION enforcements. For example, PRINTDISPOSITION=CLOSE causes a request to be 'complete' when a program closes the print file whereas a setting of EOJ means that the request, possibly holding multiple print files, will only be released once the program has gone to end-of-job.

As with the other LOG contexts, LOGPS programs can be activated with the WHEN command or used to scan one or more Sumlogs using EVAL. It should be noted that the Supervisor PRINTS context, which is only permitted with EVAL, does not access the Sumlog. Instead, PRINTS context programs can be used to scan the current PrintS request queue (as seen in the response to the ODT command PS SHOW).

In the following example, this SITUation is attempting to capture a request completion notice for a specific job and usercode and then automatically modify the FORMID and DESTINATION for subsequent printing.

```
TT DEFINE + SITUATION MYPS(LOGPS=COMPLETION):
   USERCODE EQL "META" AND JOBNAME INCL "FAMILYMANAGER" AND
   FORMID NEQ "SPECIAL"

TT DEFINE + ODTSEQENCE MYPS(LOGPS):
   ODT("PS MODIFY ",REQUESTNO, " FORMID = ""SPECIAL"",",
                  "DESTINATION = ""LP4""");
   ODT("PS FORCE ",REQUESTNO);

TT WHEN MYPS DO MYPS
```

MESSAGE context

The MESSAGE context is well-known to Supervisor users; indeed, this context is often one of the primary reasons that customers buy the product because of its effective and flexible handling of potentially critical systems messages. Actually, MESSAGE is another LOG context capable of capturing all system and display messages written to the SUMLOG (Major type 14).

Because MESSAGE is now a LOG context, this means that the EVAL command can be used to search for specific messages in one or more Sumlogs. See the section **LOG contexts and the EVAL command** for more information

LOG context

The LOG context is the generic option. Its use allows the capture of all log records as they are written to the SUMLOG by the system.

```
DEFINE + SITUATION LOG_VOL(LOG):
   LOGMAJOR=15  AND LOGMINOR=5

DEFINE + ODTSEQUENCE LOG_VOL(LOG):
   SHOW(LOGTEXT)

TT WHEN LOG_VOL DO LOG_VOL
```

The above SITUation would capture all SUMLOG records but discard all records other than Log Volume entries (Majortype=15) and Tape Volume Newfile (Minortype=5). The ODTSequence performs a SHOW of the LOGTEXT attribute, which will return an exact LOGANALYZER representation of the specified entry (as seen by the LOG VOLUMES variant). This is not a cheap option; the capture of every log record by a Supervisor WHEN on a busy system may be a significant overhead.

From Supervisor version 42.420.70, a means is available to avoid this problem. LOG context SITUations can now take a numeric list as a subcontext, allowing the user to associate a specific Major type and/or multiple Minor types. For example, a SITUATion to detect LOG Major type 15 (Volume Status entry) would be:

```
TT DEFINE + SITUATION LOG_VOL(LOG=15):
   TRUE
```

However, this SITUation would capture all valid minor types 1, 2, 3, 4, 5, 12 and 13. If only minor types 5 was required

```
TT DEFINE + SITUATION LOG_VOL(LOG=15,5):
   TRUE
```

Only one Major type may be specified per SITUATION and it must be the first number in the list. SUPERVISOR will check that any supplied Minor types are valid. Adding other minor types is done by extending the list:

```
TT DEFINE + SITUATION LOG_VOL(LOG=15,5,3):
```

would additionally capture minor type 3 (Tape Volume Purged).

Note that the LOGTEXT attribute must be used to examine the content of the LOG record returned; as stated earlier, this special string attribute represents the data seen in a LOGANALYZER report for the particular record. Also, the LOGDAY and LOGTIME attributes provide the time and date that the log record was created on the system. The DATETOTEXT and

TIME functions are useful for converting these values into meaningful values. For example:

```
DATETOTEXT(LOGDAY, MMDDYYYY)  might return  "12/01/1996"
TIME(LOGTIME)                 might return  "10:22:36"
```

Quick guide to getting LOG context help

Some hints on how to get help for the LOG context implementation.

- Use the **TT PRINT ATTRIBUTES** command from a Supervisor window to get a complete print-out of each attribute subset and the OPAL program contexts to which they belong.

- Get information about all the attributes associated with an individual context using the **TT HELP ATTR =:<context>** command:

```
TT HELP ATTR =:LOG
TT HELP ATTR =:SECURITY
```

- Check SUPERVISOR's on-line help for the following commands:

```
TT HELP EVAL
TT HELP WHEN
```

- The **EVAL** command, in particular, has extensive syntax to allow access to the current and offline SUMLOGs, with comprehensive date and time support, plus flexible controls to limit searching.

- Pay particular attention to the generic LOG attributes (use **TT HELP ATTR =:LOG** to see its attaribute subset). These may be used in any other log context and can be used to trap information for every SUMLOG record. In particualr, the LOGTEXT attribute returns a string which simulates the output from the LOGANALYZER program. This means that it is very easy to capture and interrogate log records which currently do NOT have their own context.

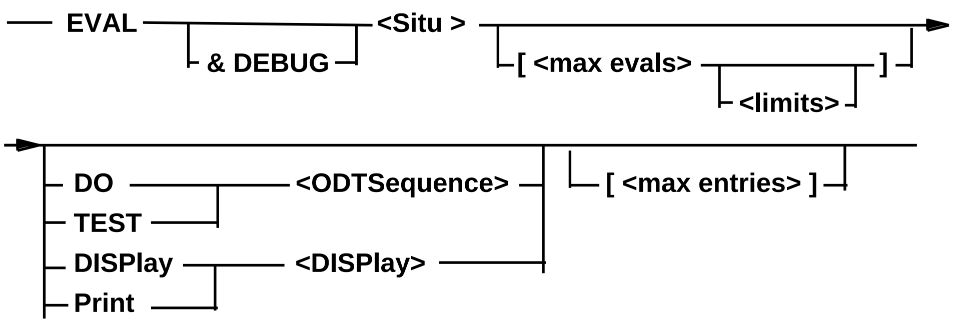
- Load some example LOG Opals directly using the ENTER command from a COMS Supervisor window.

```
ENTER DEF LOG= FROM OPALS/SUPERVISOR/EXAMPLES
```

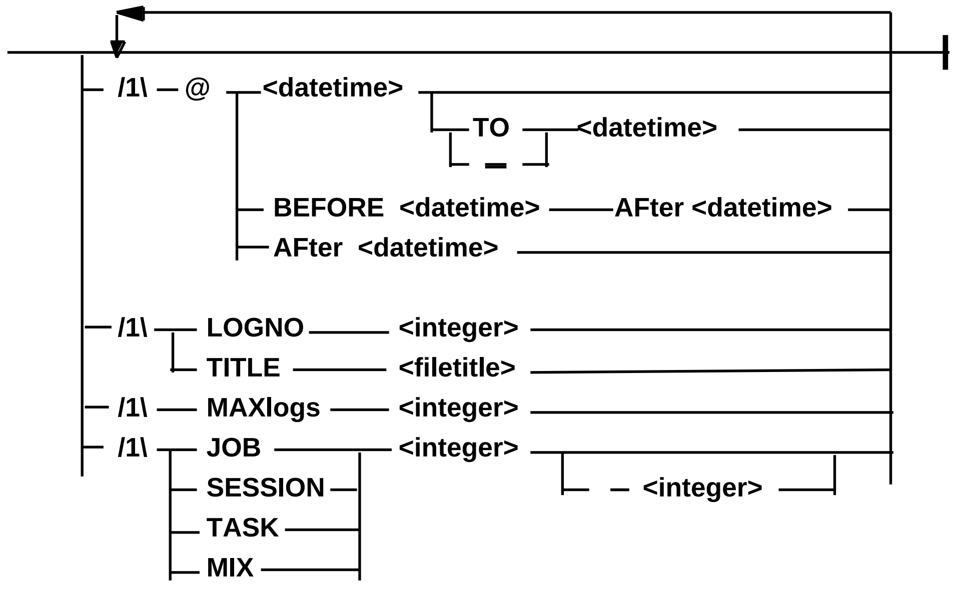
- As usual, this file may be loaded from the Metalogic release tape, if it is not already available. These simple OPAL programs give some idea on how to use the various new contexts.

LOG contexts and the EVAL command

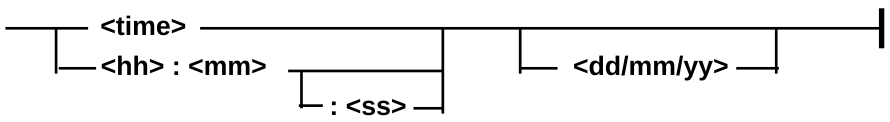
The **EVAL** command has been considerably enhanced to support the new LOG contexts, though some of the new functionality is available for all OPAL contexts. The revised syntax is shown below:



<limits>



<datetime>



The extended syntax for the **EVAL** command is discussed in some detail over the next few pages. Consider a simple SITUation and ODTSequence to handle the scanning of Sumlogs for, say, BOJ and BOT records:

```
TT DEFINE + SITUATION  BOJ_SCAN(LOGBOJ):
     USERCODE = "META" AND NAME INCL "SYSTEM"

TT DEFINE + ODTSEQUENCE BOJ_SCAN(LOGBOJ):
     SHOW(DATETOTEXT(LOGDAY,DDMMYY),,TIME(LOGTIME),,LOGTEXT);

TT EVAL BOJ_SCAN DO BOJ_SCAN
```

From a Supervisor window, typical output might be:

```
16/09/96 16:16:27 2154 *SYSTEM/DCALGOL
16/09/96 16:15:13 2152 *SYSTEM/PATCH
16/09/96 16:10:27 2149 *SYSTEM/XREFANALYZER
16/09/96 16:04:27 2130 *SYSTEM/DCALGOL
16/09/96 16:03:27 2122 *SYSTEM/DCALGOL
16/09/96 16:00:27 2118 *SYSTEM/DCALGOL
```

The above **EVAL** statement would scan the current *SYSTEM/SUMLOG file, searching for all BOJ or BOT log records and applying the SITUation check on each entry found. The LOGANALYZER utility has the ability to scan individual SUMLOG files and to refine search parameters by allowing the provision of start and end date/times. With the new **EVAL** syntax, Supervisor now has the capability to provide this kind of filtering and considerably more.

When the above **EVAL** command is invoked, Supervisor will allocate a slot to the activity as normal but it also invokes a son task, called EVREADER:

METALOGIC/SUPERVISOR/EVREADER

which will remains active whilst the **EVAL** is running. This EVREADER task is responsible for reading the SUMLOG file directly rather than imposing a heavy overhead on the main Supervisor stack.

When multiple, sequential **EVALs** are invoked whose contexts are TAPEDB, MSG or any of the LOG contexts, and the Supervisor option **LOGMINIMAL** is set, a new SUPERVISOR process called:

METALOGIC/SUPERVISOR/EVREADER/SHARED

will be automatically invoked to help reduce the system overhead of these activities. This process will remain in the mix for up to 15 minutes after its last usage.

Controlling the number of EVALuations

This feature allows the user to halt an **EVAL** after processing a certain number of evaluations. When specified using the **`<max evals>`** integer field, the **EVAL** will be automatically stopped if the current maximum number of evaluation exceeds the user limit. For LOG contexts, the default maximum of evaluations for an **EVAL** command is 65535 otherwise it is infinite. For example,

```
TT EVAL BOJ_SCAN [100] DO BOJ_SCAN
```

```
SUPERVISOR/BOJ_SCAN:RETURNED TRUE FOR 8 RECORDS:Maxevals limit reached
```

When the BOJ_SCAN has processed 100 BOJ records, the **EVAL** will stop. Note that this does not necessarily mean that there were 100 entries into the BOJ_SCAN ODTSequence.

This syntax is permitted for all SITUation types.

Controlling the number of ODTSequence entries

Similarly, the number of ODTSequence entries can be controlled using the **<max entries>** integer field. This specifies the maximum number of times the specified ODTSequence or DISPlay can be entered before the WHEN slot is terminated. For LOG contexts, the default maximum of evaluations for an **EVAL** command is 65535 otherwise it is infinite.

For example:

```
TT EVAL BOJ_SCAN DO BOJ_SCAN [5]
```

```
SUPERVISOR/BOJ_SCAN:RETURNED TRUE FOR 5 RECORDS:Maxdo limit reached
```

Here, the EVAL will stop once the ODTSequence has been entered five times, regardless of the number of evaluations.

This syntax is permitted for all SITUation types.

Limits can be imposed on both if needed. For example:

```
TT EVAL BOJ_SCAN [100] DO BOJ_SCAN [5]
```

would stop the **EVAL** once 100 BOJs had been handled or if 5 BOJ records passed the SITUation check and entered the ODTSequence, which ever comes first.

Date and time controls

In general, the searching of SUMLOGs will be limited to the current system SUMLOG (or that specified via the **TITLE** modifier) unless any of the **<datetime>**, **BEFORE**, **AFTER, MIX, TASK, SESSION** or **MAXLOGS** modifiers have been used. This section discusses the usage of the **<datetime>** modifier in some detail.

This feature is very similar to that of LOGANALYZER except for several important aspects. First, the provision of date and/or times is free-format; if a start date is given you do not need to specify a start time or an end date or time. Secondly, providing SUMLOG names which match the date and time range is not necessary since Supervisor will **automatically** search the system for the relevant files.

The value of the <dd/mm/yy> date field depends on the setting of the **USDATES** option, which is controlled via the **TT SO** command. The <yy> field may be 2- or 4-digit years. Seconds are permitted in the time specification which may include ':' delimiters if desired. For example:

```
TT EVAL BOJ_SCAN [@11:00:02] DO BOJ_SCAN
TT EVAL BOJ_SCAN [@1100-1430] DO BOJ_SCAN
TT EVAL BOJ_SCAN [@14:30-11:30] DO BOJ_SCAN
TT EVAL BOJ_SCAN [@11:00 12/1/96] DO BOJ_SCAN
TT EVAL BOJ_SCAN [@BEFORE 11:00 AF 08:00] DO BOJ_SCAN
TT EVAL BOJ_SCAN [@BEFORE 11:00 12/1/96] DO BOJ_SCAN
```

If no date field is specified and any of the times specified are greater than the current time, then yesterday's date will be assumed; otherwise today's date will be used. Dates or times entered with **"-"** or **"TO"** are order independent i.e. the lesser date/time of the range can be either first or last.

If a start date field only is provided then Supervisor will assume the end date as being the current date and time.

Controlling access to SUMLOG files

When scanning for log record entries, previous SUMLOGs are automatically searched if they are included in the given time range. Supervisor searches for SUMLOG files on various locations according to the following precedence:

- The family specified for the SUMLOG name given by the **TITLE** modifier, if present.
- The family specified by the **DL LOG** command.
- The family specified by the **TT USE FAMILY FOR LOGS** command.

For example,

```
TT EVAL BOJ_SCAN[TITLE=*SUMLOG/6343/050196/000756 ON PACK @0900 5/1/96
                                            DO BOJ_SCAN
```

The above entry would search the given SUMLOG for BOJ records from the specified date and time up to the current date and time. Assuming that the **DL LOG** family is DISK and the **TT USE FAMILY FOR LOGS** specification is WORK, then Supervisor would initially search PACK for the specified SUMLOG file. If found, the SUMLOG file is processed and PACK will again be searched, if necessary, for other eligible logs. If the specified SUMLOG is NOT found , then Supervisor will search DISK and then WORK. Again, if no eligible SUMLOGs are found on PACK, the same two families will also be scanned.

Supervisor requires that the SUMLOG files, wherever they might reside, to have consecutive log sequence file numbers. If a missing file is encountered, the **EVAL** will automatically stop.

The **MAXLOGS** modifier restricts the maximum number of SUMLOGs that will be searched by Supervisor for the **EVAL**. If the **MAXLOGS** modifier is not used then all available SUMLOG files are eligible for searching.

```
TT EVAL BOJ_SCAN{MAXLOGS=3 @0900 23/5/1996] DO BOJ_SCAN
```

In the above example, Supervisor would search the **DL LOG** family and **USE FAMILY FOR LOGS** family for all SUMLOGs that have information for the specified time range. However, in this case, only three SUMLOGs in total will be searched starting with *SYSTEM/SUMLOG, which is always accessed first. If the **MAXLOGS** limit is exceeded then the **EVAL** will be terminated with the message similar to:

```
SUPERVISOR/BOJ_SCAN:RETURNED TRUE FOR 8 RECORDS:Maxlogs limit reach
```

It should be noted that SUMLOG titles must be consistent if they are to be found automatically by Supervisor. These file tiles should have the following standard format:

```
SUMLOG/<system serial no>/<mmddyy>/<log sequence no>

e.g. *SUMLOG/6343/120196/001235
```

When a **TITLE** modifier has been specified, any leading string may be in the title before 'SUMLOG/' but all prior logs must have the same name prefix. That is, if the specified SUMLOG was held under the directory (META)MYSYSTEM/SUMLOG/.... then Supervisor will expect all other eligible SUMLOG files to be held in similar directories. However, Supervisor will always expect current log file to be entitled *SYSTEM/SUMLOG, regardless of where other SUMLOGs may reside.

The **<log sequence number>** level in the number is a 6-digit internal file sequence number maintained only by the MCP. Whenever the current SUMLOG file is closed, either by a **TL** command or because it has been automatically closed by the system due to its size, this sequence number is incremented by 1.

BEFORE and AFTER controls

BEFORE and **AFTER** modifiers are alternative way of specifying start and end date/time ranges. If you remember that **EVALs** always read the SUMLOG backwards, a **BEFORE** date/ time specification will become the start time for the scan. This starting point is found by opening the first SUMLOG file specified, or the current system SUMLOG. If this SUMLOG's starting date/time is prior to the **BEFORE** time, the next most recent SUMLOG is searched for and checked. This process will continues until the correct SUMLOG file is found or all the available SUMLOGs file have been exhausted.

When the relevant SUMLOG is found, a binary search is employed to locate the last record whose time (truncated to the nearest second) that is less than or equal to the specified **BEFORE** time.

Log records are then searched, looking for the specified Major and Minor log types (as given by the context of the SITUation), until a record is found whose timestamp is prior to that provided by the **AFTER** modifier. At this point, the **EVAL** will terminate. If no

It should be noted that any date and time changes in any of the scanned SUMLOG files will be ignored. Also, an arbitrary time of 59 seconds is given to a **BEFORE** time where the seconds field is absent. This allows the EVAL to include the scanning of all log records that "belong" to that minute.

Some examples:

```
TT EVAL BOJ_SCAN[@BEFORE 11:00 12/8/96] DO BOJ_SCAN
TT EVAL BOJ_SCAN[@BEFORE 1400 13/8/96 AF 1300 11/8/96] DO BOJ_SCAN
```

In the first example, Supervisor will search for the first SUMLOG, starting with the current system SUMLOG, which has the records for 11:00am on the 12th August 1996. Once this date/time is found, Supervisor will scan backwards from that time traversing other SUMLOGs that are available on the system, stopping only when the there are no more SUMLOGs available or **MAXLOGS** has been exceeded.

Using task numbers for search controls

MIX, **JOB**, **TASK** and **SESSion** selection is also permitted in the specification of **EVAL** limits as a means of controlling the searching of SUMLOGs. Starting from the current system SUMLOG file, Supervisor will search all log records, passing relevant entries to the SITUation, but stops as soon as a BOJ, BOT or session log record is found that matches the selection provided.

If a mix number range is provided then this behaves in a similar way to the **BEFORE** and **AFTER** modifiers; Supervisor will use the higher mixnumber in the range as a **BEFORE** modifier and the lower value will behave as an **AFTER** specification.

For example:

```
TT EVAL BOJ_SCAN [MIX 3133] DO BOJ_SCAN
TT EVAL BOJ_SCAN [MIX 3133-3158] DO BOJ_SCAN
```

In the first example, Supervisor will scan back from the end of the current SUMLOG searching for a BOJ/BOT or session entry matching mix number 3133. As soon as this record is seen, the **EVAL** will terminate. In the second example, Supervisor will begin analysing log records as soon as a BOJ entry is seen for mix number 3158, scanning back through the log from this point as normal. If a task entry for mix number 3133 is subsequently encountered, the **EVAL** will then terminate normally.

Interrogating and stopping EVALs

For very long EVAL requests that may search multiple SUMLOGs, Supervisor is able to interrogate the current status using the **EVAL** command. Each EVALREADER task that is running has special in-built pacing code that releases control of the CPU after reading 50 log records. This mechanism allows the main Supervisor stack to regain control at regular intervals (along with other active tasks).

```
TT EV BOJ_SCAN [4000 @1100 12/9/96] DO BOJ_SCAN

TT EV ? BOJ=

         ----- SUPERVISOR WHEN STATUS (LIMIT = 40, ACTIVE = 14) ---
W 067 3181 EVAL BOJ        DO   BOJ        RUNNING
                             TIMES:CPU=00:00:02,IO=00:00:00,ET=00:0
     201 evals Limit 4000 After 11:00 12/09/96 @ 16:24:01 17/09/96
     (90% of #000787), 4 ODTS entries
```

In the above example, the **EVAL** has been presented with a limit of 4000 evaluations and a finishing time of 11:00am on the 12th September 1996. In the interrogation shown above the BOJ_SCAN **EVAL** has processed 201 BOJ/BOT log records and is currently at 16:24 on the 17th September. SUMLOG log sequence number 000787 is currently being processed and 90% of that file has been processed. Only 4 entries to the ODTSequence have been processed.

```
TT EV BOJ_SCAN [1000 @BEFORE 12:00 15/9/96] DO BOJ_SCAN

TT EV ? BOJ=

         ----- SUPERVISOR WHEN STATUS (LIMIT = 40, ACTIVE = 14) ---
W 067 3183 EVAL BOJ        DO   BOJ        RUNNING
                             TIMES:CPU=00:00:04,IO=00:00:00,ET=00:0
     401 evals Limit 1000 Before 12:00 15/09/96 @ 14:19:07 14/09/9
     (17% of #000785), 6 ODTS entries
```

Here, the **EVAL** uses a **BEFORE** modifier to set up a start date/time control but this time a default evaluation limit of 1000 has been assigned.

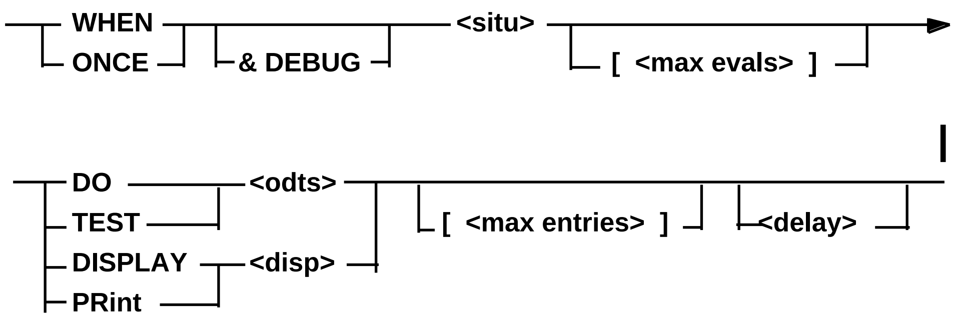
Any LOG context EVAL can be terminated in the usual manner whilst processing SUMLOG file records:

```
TT EVAL BOJ_SCAN DO
```

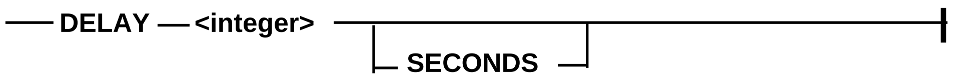
On occasion, it may take some time Supervisor to honour a termination request especially if there is a large number of messages, generated by the **EVAL**, are queued to the originating station. In such cases, the EVALREADER process may have terminated normally, having processed all relevant SUMLOG files, by the time Supervisor gets the opportunity to process the deactivation request.

LOG contexts and the WHEN command

The **WHEN** command has been similarly enhanced to limit the number of evaluations and ODTSequence/DISPlay entries:



<delay>



The **<max evals>** and **<max entries>** modifiers have already been discussed in the previous section and for the **WHEN** command, the effects are identical. The **WHEN** will automatically be stopped when either of the limits are exceeded.

By default, there are no practical limits on number of evaluations and ODTSequence entries that a **WHEN** may perform. All other modifiers permissable for **EVAL** are not allowed with the **WHEN** command.

Useful information about LOG contexts

As with the TRIM component of Supervisor, LOG contexts is protected by a software license key. If this key has expired or does not exist, any attempt to DEFINE or execute a LOG context OPAL program will return the error message:

```
NEED LOGCON KEY
```

In such cases, you should contact your nearest Metalogic site for further support.

When the Supervisor options, **LOGMINIMAL** and **MONITORING**, are set and reset respectively, Supervisor will set the FILEACCOUNTING task attribute to ANONYMOUS for the following tasks:

```
CONTROLCARD
EVREADER
FILEDATA
LOGANALYZER
CHECKWFL
```

These settings can significantly reduce some of the system overheads in accessing various SUMLOG files. Note that the ODT ACCOUNTING command must be set as follows:

```
ACCOUNTING  FILE = UNSPECIFIED
```

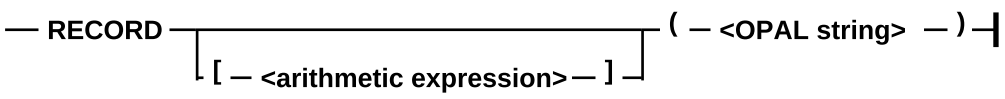
Using HOTLINE and RECORDER

Although SUPERVISOR makes the best use of standard A Series resources for reporting (messages to system consoles or REMOTESPOs, log entries, printouts), there are times when it may be desirable to stream information to one or more user-specified destinations in a customised format. To this end, METALOGIC have implemented the HOTLINE and RECORDER mechanisms.

Within an OPAL program, it is possible, by use of the RECORD verb, to direct output to a specific disk file. In addition, the information can be directed to a BNA Port File interface. This means, for example, that critical system events can be detected and alert messages directed through BNA Port Files to a non-A Series environment. Some SUPERVISOR users have implemented automatic operator alert systems using this mechanism to redirect OPAL output to IBM PC/Windows-type environments; others have linked into paging systems so voice messages can be generated and telephoned to off-site personnel.

METALOGIC provides sample, usable RECORDER and HOTLINE programs in source form on the software release tape. The purpose of this section is to document the user interface to the HOTLINE program, and give guidelines for customizing both programs.

<RECORD Statement>



The OPAL verb RECORD passes a string from an ODTSequence to a program called the RECORDER, generally for the purpose of logging some condition to a disk file or passing information to a HOTLINE program. RECORDER and HOTLINE are designed to be user-customizable. The following describes the action of RECORD if used with the standard RECORDER and HOTLINE programs. All these actions can be customized.

Refer to the **OPAL Programming Reference Manual** for more details on the RECORD statement.

Logging to a disk file

RECORDER creates up to six disk files, 0 through 5, at one time. To log to files 1 through 5, enter the file number in brackets and the text to be logged in parentheses. For example:

```
RECORD [3] (TIMEDATE(YYYYMMDDHHMMSS),,”User”,,USER,,
   “program”,,NAME,,”security violation (COMPSEC)”);
```

RECORDER does not automatically log either the time and date, or the identification of the originating OPAL. Include these in the text (as shown in the example) so that you can identify the messages. This is especially important if several of your OPALs record to the same file number.

Logging to disk file 0

Use exactly the same OPAL to log to file 0 as to the other files, but with the file number = 0. However, RECORDER treats file 0 specially. An extra line showing the time of day and identifying the originating OPAL will be added by RECORDER. The BLOCKSIZE of the file will be 450. The file will be closed after each log entry. If the log string begins with “**QUIT**”, RECORDER will go to EOT after logging the entry.

Description of the files created

The files are named ***SUPERVISOR/RECORD/<dayname>/<filenumber>**. **<dayname>** is the day of the week that the file was created (**MONDAY, TUESDAY,** etc.). The files are placed on the system DL BACKUP family. New files are *not* automatically created at midnight. If you want new files for each day – which is a good idea – then schedule an OPAL to run at midnight every day which causes RECORDER to quit (see below). SUPERVISOR will automatically re-initiate RECORDER, thereby creating new files for each day.

The files are created as standard JOBSYMBOL files, except that for files 1 through 5, BLOCKSIZE=30. All files are opened with PROTECTION=PROTECTED, which in combination with the blocksize and a single buffer means that at most two records can be lost due to a system failure. File 0 is created with MAXRECSIZE=450, but no records will be lost, because the file is closed after each record is written.

Logging to disk file 11

A special case is recording to file number eleven (11) when RECORDER will both send messages to HOTLINE and log them to a disk file. The file is not named as described above but instead has the general form:

***SUPERVISOR/RECORD/<dayname>/HOTLINE**

This allows a site to keep a record of messages sent to HOTLINE.

Causing RECORDER to quit

Sending a message starting with “**QUIT**” to file 0 will cause RECORDER to quit. Supervisor will automatically re-initiate RECORDER the next time a RECORD is executed. In normal operation, the only reason to do this is to force RECORDER to create new files immediately after midnight (see “**Description of the files created**”, above). For example:

```
TT DEF ODTS QUIT_RECORDER: RECORD [0] (“QUIT”);
TT AF 0000 DAILY:DO QUIT_RECORDER
```

Sending to the HOTLINE program

Use the same OPAL syntax as for logging to a disk file (see above). Use a file number (here called a *message class*) of 6 to 47. RECORDER will send the message to any HOTLINE program which has requested that class of messages.

The supplied HOTLINE program can be run from any terminal. It asks the operator what hosts, and what message classes, to monitor. The operator can select message classes 6 through 9 singly, or can select all message classes (6 through 47). HOTLINE then connects to the RECORDER program on the selected hosts and requests the selected message classes. Upon receiving the messages from RECORDER, it writes the messages to the terminal.

If no HOTLINE program is currently requesting the message class, the message is discarded. If the message class is invalid, the message is discarded.

Messages displayed by HOTLINE include time and host identification. If you need to identify the specific source OPAL, you should write the RECORD statement to include this information.

Operating the HOTLINE program

Operating the HOTLINE program is a simple matter of running the program and answering a couple of questions. From CANDE, enter

```
RUN *METALOGIC/SUPERVISOR/HOTLINE
```

HOTLINE will ask whether you want all messages, or just one class. After you answer, it will ask you which BNA hosts you want to monitor. You need not enter the local host name; the local host is always monitored. Enter the names of any other hosts you want to monitor, one name per transmission. Then transmit **?END**.

HOTLINE will then enter monitoring mode, displaying each message on your terminal as the message arrives. Enter **?HI2** at any time to receive a status report on currently connected hosts. Enter **?HI1** to terminate HOTLINE.

Multiple copies of the HOTLINE program can be running at the same time, either on the same host or on different hosts. If more than one copy of HOTLINE requests the same message class, all copies of HOTLINE receive copies of all messages in that message class.