TECHNICAL MEMORANDUM

86

SMS: Supervisor Monitor Scheduler

F. Hollmann

Operations Department

January 1984

This paper has not been published and should be regarded as an Internal Report from ECMWF.
Permission to quote from it should be obtained from the ECMWF.

European Centre for Medium-Range Weather Forecasts
Europäisches Zentrum für mittelfristige Wettervorhersage
Centre européen pour les prévisions météorologiques à moyen

1 ACKNOWLEDGEMENTS

Thanks go to Claude Guillerand* who worked with me on the project and who gave me an insight into Meteorology.

Frank Hollmann.
August 1983.

* Version I of the sub program was designed and implemented by C. Guillerand and T. Stanford and ran operationally from August 1979 until April 1981. A rewritten version II was produced by C. Guillerand and F. Hollmann, and has run operationally since April 1981, with extensions by F. Hollmann in December 1981.

2 Preface

This report is intended to give a description of ECMWF's forecasting system, the concepts employed at the centre for running operational weather forecasts and a description of Supervisor-Monitor-Scheduler (SMS) which can be considered the heart of the whole forecasting suite. It should be noted already at this point that, although originally planned for ECMWF's forecasting suite, SMS is general purpose.

For the more technical reader a summary of operator capabilities and SMS interaction commands follows. The paper will also give information on the technical implementation of SMS and certain installation notes.

3 Suite Concept For Operational Forecasts At ECMWF

The basic flow of meteorological information at ECMWF is similar to the flow of information in any meteorological centre in the world. ECMWF's Meteorological Operational System (EMOS) can be sub-divided into a number of logical sub-processes like data acquisition, pre-processing, analysis, forecast, post-processing and dissemination. Furthermore real-time operational aspects are taken into account as well as graphical display of forecast results and a systematic archiving. Many of these processes are asynchronous while others are order-dependent and have to be synchronised. This basic scheme lead to the definition of a suite.

3.1 Definitions: Suite, Family, Task, Event

The term suite is used for a logical structure describing any kind of processes (synchronous and asynchronous). A suite is characterised by its suite type which is a single letter.

Any such suite will consist of an ordered set of families as the next smaller logical entities. A family is a logical structure describing the inter-relation of processes belonging together.

In general a family will consist of a non-empty set of tasks with various attributes (task attributes) and a possibly empty set of other families (with event attributes) which all belong logically together. In this context 'processes belonging together' now means that if $task_L$ of $family_F$ is executing then $task_{L+1}$ .. $task_N$ of $family_F$ will be candidates for execution sooner or later and if $family_F$ contains other families with event attributes these families will be candidates for executions sooner or later as well. Every family within a suite is defined by its 5-character family name, the first character being the suite type followed by another letter and three digits. Within a suite family names must be unique.

A task now is considered a batch job to be executed on some computer in the local computer network. A task can be defined with certain attributes (as explained later) to give enough flexibility; a task is identified by its 5-character task name, the first two characters being equal to the suite type, followed by another letter and a 2-digit sequence number. Within a family task names (and task sequence numbers) have to be unique but tasks in different families may have the same name.

The event in the concept is being used as a means of synchronising the executions of families and tasks. Events can be set by tasks after they have reached certain steps and are used to 'trigger-off' new tasks in the same family

and to execute other families. It is important to note that the event is an entity of a single family only, eg. an event set by a task of one family cannot be used to trigger-off a task in a different family; but it is only possible to trigger-off other families. This being necessary as the successful run of two different families might require to run a third family, where the intonation is on ´successful´, which otherwise would be impossible. An event is characterised by a 5-character numerical quantity. The first two digits have to match the last two digits of one task within that family where the event is to be used while the last three digits are arbitrary event sequence numbers. Within a family events have to be unique.

3.2 Family Definitions

3.2.1 ´START´, ´END´, ´P´

Various times can be specified for a family like start-times and end-times which will allow the execution of families at pre-defined times and/or only during pre-defined intervals or only prior to a specified end-time. It is also possible to specify periodic restart times in order to allow for applications which have to run repeatedly. All three specifications are usable in any combination.

3.2.2 ´TRIGGER´

The TRIGGER keyword is used in a family definition if that family is to be triggered by event(s) issued by tasks from other families. The non-empty list following the keyword must consist of pairs containing the family name of the issuing task and the event itself (note that this definition uniquely identifies one single task). This concept allows to link executions of different families in a fairly complex way.

3.2.3 Equivalenced Families (<=>)

In cases of families equivalent to other families it is possible to simply define: $family_a$ <=> $family_b$ and omitting the family body (see below) for $family_a$. Of course, $family_b$ has to be defined (somewhere) within the suite. Timers and triggers as defined above can be different for these families if any.

3.2.4 ´TASK´ And ´FAMILY´

The TASK and FAMILY keywords are used to define the task and families, if any, with their attributes as explained earlier.

3.3 Definitions For The Family Body

3.3.1 ´EVENT´

The ´EVENT´ keyword can be used for tasks and families in the family definition - followed by a non-empty list of events - to define their inter-relation. Any such event is expected to be set by some other task within the family (note: events are unique within one single family). A task or a family is to be triggered-off once all events have been set.

It follows from this approach that at least one task has to exist with no event attribute.

3.3.2 'TIME'

In order to be able to start a task within a family at a particular time the 'TIME' keyword followed by a start time can be specified for any task. This mechanism allows for even more flexibility as compared just with the time attributes of the family definition alone.

3.3.3 'DAY', 'DATE'

As certain tasks are expected to run only on pre-defined days each week or at pre-defined dates within each month provision is made for that by use of the 'DAY' parameter and 'DATE' parameter respectively which have to be followed by a list of at least one day or one date.

'DAY' and 'DATE' cannot be used together.

3.3.4 'PRINT'

By default outputs of successful tasks initiated by the supervisor will not be printed. To be able to receive the line printer listing of a task the 'PRINT' keyword can be specified.

Note that all outputs will be catalogued by SMS on the default permanent file base for subsequent post-processing and that for all failing tasks and for all tasks with a 'PRINT' parameter outputs are also routed to the central line printer.

3.3.5 'NORESTART'

Experience has shown that the majority of all failing tasks will run to completion on the second attempt. Thus, in case of a first failure of a task, SMS initiates a re-run automatically. On the other hand there are applications which must not be re-run. To prohibit automatic resubmission of tasks the 'NORESTART' parameter must be specified. (The interested reader is referred to the SMS Summary: Restart processing).

3.3.6 Machine Type Definition

SMS has been designed to work in a multi-mainframe environment consisting of a maximum of two CYBERs and a maximum of two CRAYs. In order to run a suite efficiently in such an environment a machine type attribute must be defined for each task to indicate on which mainframe in the network it is supposed to run. It is also possible to define certain tasks as candidates for load levelling if machines are available with that feature enabled.

The following keyword type parameters can be defined:

CYBER To run on any CYBER.

CY1 To run on a CYBER with a logical ID corresponding to CY1.

CY2 To run on a CYBER with a logical ID corresponding to CY2.

CRAY To run on any CRAY (if more than one).

CR1 To run on a CRAY with a logical ID corresponding to CR1.

CR2 To run on a CRAY with a logical ID corresponding to CR2.

(CYBER, CY1, CY2, CRAY, CR1, CR2 have to have corresponding logical mainframes IDs. This is explained in more detail in the SMS Command Summary: INGEN - Initialisation File Handler).

3.4 Examples Of Family Descriptions

A small context-free language has been developed to define a suite. The attributes and keywords defined in the previous paragraphs are actually terminal symbols (keywords) in that language. The syntax is free-format, so it is up to the user to write his suite definitions in a ´readable´ form. Commas and blanks are valid separators. Any family must be defined starting with a family name and terminating with a ´.´.

```
OE924,
  TASK = OOE01,
  TASK = OOM06, CYBER, EVENT = 01001,
  TASK = OOK00, CRAY, EVENT = 06001, DAY = (MON, FRI),
  TASK = OOL35, CY2, TIME = 17:15, DATE = 15, NORESTART,
  FAMILY = OK224, EVENT = 01001.


OK224, END = 23:59,
  FAMILY = OO123, EVENT = 99001,
  TASK = OOK99.


OK955, TRIGGER = (OE924,01001), (OK224,99002),
  TASK = OO055, PRINT.


OO123, START = 04:30, P = 00:30,
  TASK = OOB51, CY1.


OK956 <=> OK955, TRIGGER = (OE924,35001).
```

4 Functional Description Of The SMS-Subsystem

The following paragraph will give a functional description of the SMS subsystem including the interface from user tasks and the operator interface.

4.1 General Considerations

The SMS subsystem has been designed for the daily, weekly or other routine work carried out on ECMWF´s computers. One main aspect was to avoid classical operator intervention. This basically means to have a supervisory system responsible for the automatic submission of jobs and with control over the running of these jobs. Of course, operator intervention must be possible in cases where the routine operation gets delayed by outside events and to not take

the operator's responsibilities away completely. Actually, the central operator must be able to change the pre-defined schedule of SMS.

4.2 How To Get Started

4.2.1 Define Suites - Task Tables

The first step, of course, is to define a schedule, eg. a suite. SMS is capable of handling up to 9 suites, the names of which can be chosen by the installation (see below for installation notes). The formal definition of a suite as explained in the previous chapter then defines the structure of the schedule with all its interactions and dependencies. This formal definition will be referred to from now on as a task table. Furthermore, every task defined in the task table has to have a corresponding job deck (with the same job name) to be invoked by SMS. These job decks have to contain valid JCL for the machines the jobs are supposed to run on. Also provision has to be made for the end-of-job/abort-job conditions (cf. SMS Command Summary: ENDT/ABTT).

It is obvious that if certain jobs are to set events (as defined in the task table) that the actual jobs have to send these events to SMS.

The TTGEN utility is provided within the subsystem which will a) syntax-check and semantic-check a task table definition, b) verify task:job-deck correspondences and c) generate the task table file for SMS if so requested. For b) and c) all job decks must be supplied as single records on a file for TTGEN.

As SMS efficiency largely depends on the structure and the correctness of these task table files, the TTGEN utility is rather a comprehensive piece of software.

4.2.2 The Initialisation File

A so-called initialisation file has to be generated as well as one of the preliminary actions. A utility program 'INGEN' (cf. SMS Command Summary) has been provided to generate, change or list the contents of an existing initialisation file. This file is to contain the proper logical mainframe IDs corresponding to the task table machine type definitions (CYBER, CY1 etc.), the terminal ID of the SMS operator Console and an optional list of up to four terminal IDs for so-called permissible jobs; eg. that also any job, not initiated by SMS, which has got a remote batch routing ID equivalent to one of these IDs will be allowed to make an SMS request.

4.2.3 SMS: Initialisation - Work Environment - Active Suites

SMS is a CYBER CPU program to be initiated by the operator at a free controlpoint (the parameter description can be found in the SMS Command Summary). SMS will first establish the work environment (also described in the SMS Command Summary) and will then go into processing mode, eg. process any queued function requests, perform internal maintenance tasks and respond to any new function request.

The central operator is given a DECWRITER terminal which functions as his operator's console, eg. the terminal serves as SMS' message logging device and can also be used for SMS commands as can be the central console. Messages from SMS are indented such that they can be easily distinguished from commands.

In order to 'start a suite' the operator must enter a 'DATE' command (cf. SMS Command Summary) to set a new date for this particular suite. This date is being copied to each task initiated by SMS later, so these tasks can get hold of it which subsequently allows to completely re-run certain suites for particular dates.

Internal processing of this command implies various checks. In particular the task table file mentioned earlier has to exist and will be processed. At the same time a so-called task table back-up file is being created which has a similar structure as the task table file but is being used for further processing to save pertinent information of the suite during execution.

Once a task table back-up file exists a suite is considered active.

During initialisation now when SMS is establishing the work environment an attempt is made to attach all back-up files of all defined suites to determine a list of all active suites. For all those suites the corresponding task table files have to exist as well and, of course, both files have to pass a consistency check.

The operator may use the 'SUITDAT' command to obtain a list of the active suite, with their corresponding dates and start times. Also the second line of SMS' B-display message contains the identification of all active suites.

A suite is removed from the list of active suites using the 'STOP,suite type' command (cf. SMS Command Summary).

4.3 The Status Of Families And Tasks

Each family within a suite has always got a certain status (Family Status) associated with it as well as each task within a family. These status flags are important because, as complexity grows, more and more possibilities arise for erroneous processing and errors invoked by operating personnel. Furthermore, only these statuses give an exact picture of the suite, given a snap shot at a particular time.

4.3.1 Family Status Flags

When a suite becomes active using the 'DATE' command all family status bits are OFF.

4.3.1.1 E-Bit (Executable Bit)

Talking in processing terms nothing will happen to a family unless that family is executable, eg. unless the E-Bit is set. The E-Bit is set using the 'EXEC' command or the 'EXST' command.

When the operator is to start a suite he will usually enter one of the above commands for a particular family or a range of families in the ordered set of families defined in the task table. From this point onwards these families are candidates for processing, but note: they are only candidates; nothing more happens.

It is therefore possible to define different sets of families within a single suite which the operator can make eligible for execution at different times.

4.3.1.2 S-Bit (Start Bit)

The S-Bit of a family will be set once this family gets started, eg. the ´START´ command has been used for an executable family or ´EXST´ has been issued for a family not yet executable (note: ´EXST´ is functionally equivalent to ´EXEC´ followed by ´START´) or in the case of certain events having occured a family was dependent upon.

To start a family will first result in a check of any timers defined for that family (eg. ´START´, ´END´ or ´P´). If there is a start time ahead of the current time an entry for this family will be made in the so-called Delay Table (which is unique to each suite on the back-up work file) and processing terminates. If there is an end time which has already been passed the family C-Bit (see below) will be set to indicate family completion (Note that in this case the A-bit (see below) is not set). In all other cases the family will be made activ (see next chapter).

4.3.1.3 A-Bit (Active Bit)

The A-Bit for a family will be set once SMS decides to make a family active. This happens in the case as explained in the previous paragraph and it also happens when SMS decides to activate a family which had been started beforehand and which had an entry in the Delay Table.

When a family is activated the family body description is read from the task table file and copied to the back-up file. Then an attempt is made to initiate all tasks with no events (but see below for ´Task Status Flags´).

4.3.1.4 C-Bit (Complete Bit)

The family C-Bit gets set, once all tasks within that family are complete (see below for task C-Bit) and if there are no events to be set from tasks within that family. The latter is important as there is no direct application-to-SMS interface which implies that an ´end-of-job´ notification might get processed before a ´set event´ notification. This becomes clear since in the case of a ´family-complete´ condition not only the family C-Bit is set but also the space occupied by the family body description on the back-up file is freed (therefore a family can only be set complete if all events have been set as well, as otherwise no family description would be present anymore).

4.3.1.5 F-Bit (Force Bit)

The F-Bit is another important status bit which will be set for a family if the operator uses the ´EXEC´ command with the ´F´ parameter. This mechanism can be employed (and should only be employed!) if a family or a range of families has to be re-run.

4.3.2 Task Status Flags

Each task within a family has certain statuses associated with it. Once a family becomes executable, eg. the family E-Bit is set, all of the task status flags are OFF.

4.3.2.1 S-Bit (Start Bit)

The S-Bit of a task will be set once a task is to be initiated. This is the case for an active family (family A-Bit set) and no events have to be set for a task.

If a task is to be initiated it will first be checked if that task has been defined with a 'DAY' parameter or a 'DATE' parameter. Should these parameters indicate that the task should not run the task will be set complete (C-Bit - see below). Please note that the task status will only show A-Bit and C-Bit! Furthermore, all tasks within this particular family waiting for events from this task are set complete as well, since the events will never be sent. These tasks then will only show the C-Bit. Processing then continues to set all families complete which are dependent on an event from the task mentioned above. These families in turn will only have the E-Bit and the C-Bit associated with them. The following example will highlight this fact:

```
0A001,
  TASK = 00A01  DATE = 12
  TASK = 00A02  EVENT = 01001
  FAMILY = 0A002  EVENT = (01001, 02001).

0A002,
  TASK = 00A03.

0A003, TRIGGER = (0A001,01001)
  TASK = 00A04.
```

Then type at the operator console: 'EXST,0A001,0A003.'

(Note: this command is equivalent to: 'EXEC,0A001,0A003.'
'START,0A001.')

The above command will set the E-Bits for families 0A001..0A003 and will start 0A001 (S-Bit set and subsequently the A-Bit). Only on each 12th of a month task 00A01 will run which in turn will invoke 00A02 and the two other families as well. On all other dates task 00A01 will not run and will be set complete followed by the other tasks and families forced complete as well. Last not least it is determined that also family 0A001 has to be set complete as all tasks in that family are complete and no family is waiting for an event to be sent from a task in this family.

If, following the above considerations, a task is a candidate for execution, it is checked if there is a start time ('TIME' parameter) defined for this task ahead of the current time. In this case an entry is made for this task in the Delay Table and processing terminates (note that only the task S-Bit is set). If the start time has been passed already or in the case of no start time the task status will be changed to active (see next paragraph).

4.3.2.2 A-Bit (Active Bit)

The A-Bit for a task will be set if this task has to be activated, eg. if the task has to be submitted for execution. This is the case as consequence of the explanations in the previous paragraph and also in the case when SMS decides to activate a task which has been started beforehand and which had an entry in the

Delay Table. (There is a further exception case for forced executions of arbitrary tasks, cf. F-Bit below).

When a task is to be activated the job file is read from the task table file (where it has been stored by TTGEN) and an input record is copied to the end of the file containing family name, task name and the date of the suite. This job is submitted to the operating system for execution and the jobname assigned by the operating system is saved in the family body description for this task on the back-up file.

4.3.2.3 C-Bit (Complete Bit)

The task C-Bit will be set, once the end-of-job notification is received by SMS. Processing of this condition also includes a check of the family-complete condition. Furthermore an entry is made in the so-called output queue table (which is common to all suites) for future use in one of the maintenance functions to attach the output file of the job and catalog it on the default permanent file base and also produce a line printer output in case of the ´PRINT´ parameter being defined for this task.

4.3.2.4 T-Bit (Abort Bit)

The T-Bit for a task will be set, once SMS receives the first task-abort notification for this task. As explained earlier one further attempt is made to run this task again unless the ´NORESTART´ parameter has been used. Upon a second failure of the job the operator is informed via his DECWRITER console that some action is required. In both cases an entry in the output queue table will be made and the outputs of these jobs will be printed irrespective of the ´PRINT´ parameter setting.

4.3.2.5 F-Bit (Force Bit)

If the operator decides to run a task independent from the normal schedule of the suite he may do so by means of the ´RUNTSK´ command (cf. SMS Command Summary). In this situation the task F-Bit will be set which flags this situation.

4.3.3 The ´STATUS´ Function

The ´STATUS´ command is provided which will give status information of the complete suite, a special family, a special task within a certain family, all incomplete families within a suite and will give information of the contents of the Delay Table. An output is produced from the back-up file for an active suite and formatted in the same way as the task table definition with the appropriate status bits indicated as well. Furthermore a list of all triggered families is provided for a family when this family appears in any ´TRIGGER´ list.

A detailed description can be found in the SMS Command Summary.

4.4 The Processing Of Suites

4.4.1 General Remarks

From the explanations in the previous sections it becomes clear that suites can be defined which will run automatically under SMS control once the operator has entered the ´DATE´ command and ´EXEC´ or ´EXST´ commands.

It must be understood that a computing environment like the one at ECMWF with two CYBERs sharing mass storage units and one CRAY connected via channel couplers to each of the CYBERs must be considered fairly loosely connected; in particular there are no general facilities for application-to-application communication within the complex. Within the standard operating system NOS/BE such an interface is provided, known as System Control POINT. This feature is being used for inter-job communication in the SMS system for all those jobs running on the same CYBER as SMS. Jobs running on the other CYBER will determine that there is no SMS running and will write their requests to a shared permanent file which will subsequently be processed by SMS. For CRAY jobs the implementation is as such that a small job is routed via the CRAY station interface into the CYBER input queue which will execute with system job priority and make an SMS request like all other CYBER jobs (cf. ´PGMFCT´: SMS Command Summary). It is important to note that these types of communication are one-way only, eg. jobs do not have to wait for an acknowledgement, a response or a return code. The situtation is a bit different for the SMS operator interface which uses bi-directional communication. That is why SMS should always run on the mainframe which also provides the INTERCOM service. It was, on the other hand, explained earlier that all operator commands can also be issued from the main console; what will be lost in this case, also assuming that the INTERCOM service is not available on the SMS mainframe, is the log provided by SMS on the DECWRITER console. For this situation the ´LOGFILE,PRINT´ command is available which will produce, when called periodically, a continous central line printer log equivalent to the terminal log.

4.4.2 ´RESTART´ Processing

As explained above SMS has only little control over jobs executing in the computer complex. It is especially impossible to get a notification of all actions a job is performing, eg. if the operating systems decide to purge a job from the system or if the central operator decides to kill a job.

A mechanism has been implemented referred to as ´RESTART´ processing which can be invoked by the operator either when he calls SMS to a control point or at any later time using the ´REST´ command (cf. SMS Command Summary). During ´RESTART´ processing SMS will sample all queues in the computer network and make sure that all of the jobs initiated by SMS do exist in one of the queues. Since this operation is very time consuming it is not executed as part of the periodically called maintenance functions but has been made an operator command. If SMS is unable to find a particular job the operator is notified and has to decide whether this task is to be re-initiated. The interested reader is referred to the SMS Command Summary where ´RESTART´ processing is described more fully.

4.4.3 SMS Operator Control

The whole idea of the SMS system is to be able to define a schedule for jobs to run without operator intervention. Nevertheless the central operator has been given full control over SMS and the running of any suite via his command terminal. Thus he is basically able to change any suite schedule. A detailed description of all operator interaction commands can be found in the SMS Command Summary.

4.4.4 SMS Interaction Commands

4.4.4.1 ´Issue Event To SMS´

As the main entity in the system is the event, every task submitted by SMS to the operating system for execution has to be able to send an event to SMS. This is either possible on the control card level or in a subroutine call. The jobs simply call-up SETEV and provide the event number.

4.4.4.2 ´Write Message To Message Logging Terminal´

A facility has been provided for jobs (initiated by SMS) to send messages to the SMS Operator Console. This is particularly useful for large suites to get a notification when certain steps have been reached. Again, a subroutine interface and a control card call have been provided for this purpose.

4.4.4.3 ´End-Task Notifications´

All jobs initiated by SMS have to issue a notification of successful completion or of unsuccessful completion as well. Two control cards have been provided (´ENDT´, ´ABTT´) have been provided to serve this purpose. They must be placed at the end of each job.

5 Operator Capabilities And SMS Interaction Commands

This chapter defines a minimum set of operator capabilities with the SMS system. These are to allow the central site operator and the command terminal operator (who can be the same) to status and control SMS and the running suite(s). The chapter also defines all other interaction commands plus some special programs for initialisation and analyses.

The following chapter is organized as follows:

-- Console capabilities.

-- SMS operator commands.

-- Other SMS commands.

-- Special programs.

Commands mainly to be used for SMS control are marked as such (´***´).

Where applicable information is also given about error messages or other messages requiring special attention by the user, although it should be noted that that list is not complete since other messages not listed here are self-explanatory.

5.1 Console Capabilities

5.1.1 SMS - Supervisor (***)

5.1.1.1 Control Card Description

```
┌──────────────────────────────┐
│ n.X SMS {parameter list}.    │      (***)
└──────────────────────────────┘
```

Initiate the supervisor at a clear control point.

The following optional non-positional parameters are allowed:

RESTART {=xyz} or

REST {=xyz} or

R {=xyz} Execute 'RESTART' function for those suites indicated by 'xyz' (each character has to be a valid suite type). In the keyword-only case the function is executed for all (installation) defined suites.

The following suites have been defined:

M: Maintenance suite.
O: Operational suite.
E: Experimental suite.
C: Accounting suite.
A: Experimental suite.
B: Experimental suite.
R: Research suite.
T: SMS test suite.

Note: Only one of the above parameters 'RESTART', 'REST' or 'R' should be used.

Default: no 'RESTART' to be performed.

CT=tid Command terminal ID. 'tid' must be a 2-character combination.

Default: command terminal ID from the initialisation file.

MT=tid Message terminal ID. 'tid' must be a 2-character combination.

Default: same as CT.

ALLMES Keyword-only parameter to send all messages to the message-logging terminal.

Default: do not send all messages.

POSLF Keyword-only parameter to 'position' the supervisor's logfile at the current EOI. This will affect the commands 'LOGANAL' and 'LOGFILE'.

Default: Logfile position will not be changed.

IDL=time — Idle down time in seconds.

Default: 5 seconds.

RCL=time — Idle recall time in milli-seconds.

Default: 0.5 seconds.

OPQ=time — Time to examine output-queue table periodically in seconds.

Default: 5 minutes.

FRQ=time — Time to re-examine request-file in seconds.

Default: 2 minutes.

HK=time — Time for various housekeeping tasks to be performed periodically in seconds.

Default: 5 minutes.

5.1.1.2 B-DISPLAY Description

The first line of the supervisor´s B-DISPLAY describes SMS´ current action while the second line holds statistics information:

```
../ct/mt   ........ .........
../ct/mt-A ........ .........
../ct/DOWN ........ .........

^  ^  ^    ^        ^       ^ suite index still loaded
|  |  |    |        | active suites
|  |  |    | remote batch IDs for permissible jobs
|  |  | message terminal ID (mt) with ALLMES indicator or
|  |  | ´DOWN´ if SMS was unable to send messages.
|  | command terminal ID (ct).
| remote batch ID for jobs running under control of SMS -
| this is the ID found in the R and H-DISPLAYs on the CYBER
| and the STATS-DISPLAY on the CRAY (installation parameter).
```

5.1.1.3 Important Messages During Initialisation

SMS initialisation is a multi-stage process and messages are ordered accordingly.

5.1.1.4 Stage I Messages - Low-Level Environment

The following error messages are produced during the first stage of the initialisation followed by an abort of the control point:

-- ERROR IN NUMERICAL PARAMETER.

-- -RESTART- PARAMETER IN ERROR.

-- ERROR IN -TID- PARAMETER.

-- NOT PROPER OVERLAY LOAD.

SMS bug: an analyst should be consulted immediately.

— CEJ DISABLED.

The ´CEJ/MEJ´ switch on the deadstart panel has to be flicked to ´ENABLE´ and a deadstart has to be performed.

-- PERMISSION CONFLICT FOR LOGFILE.

Another user has SMS´ logfile attached. This usually happens in a multi-mainframe environment if an attempt is made to bring up the supervisor on both mainframes.

-- REQUEST ERROR nnnn FOR LOGFILE.

Consult an analyst.

-- CATALOG ERROR nn.

Consult an analyst.

-- ATTACH ERROR nn FOR lfn

Consult an analyst.

-- INSUFFICIENT PERMISSIONS ON LOGFILE.

We do not have READ, MODIFY and EXTEND permissions for our logfile, eg. some other user has the file attached.

-- SETP LOGFILE ERROR.

Can only happen with use of ´POSLF´ parameter.

— NO DATA ON INITIALISATION FILE

The initialisation file is empty. Run ´INGEN´ to generate the file.

5.1.1.5 Stage II Messages - High Level Environment

The following errors are detected during stage II of the initialisation process. The errors within the first group are fatal as the integrity of the work file(s) is somehow destroyed. The messages should be acknowledged with an ´n.GO´ type-in to resume supervisor execution (without the suite in error). An analyst should be consulted immediately.

-- _.-SUITE-*STA_* FILE SIZE ERROR - FATAL

-- _.-SUITE-*STA_* CHECKSUM ERROR - FATAL

-- _.-SUITE-*STAS*/*STAB* INCOMPATIBLE - FATAL

This is almost likely a case where someone has generated a new task table source file STAS without purging the old back-up file STAB (cf. description of TTGEN).

-- _.-SUITE-*STA_* ATTACH ERROR __ - FATAL

The error code should be looked up in the SPRM and an analyst should be called if necessary.

The following error messages are related to the file-request interface.

-- lfn ATTACH ERROR __
N.CFO NEW OR ABT

An ´n.CFO´ type-in is required to either generate a ´NEW´ file ´lfn´ or to abort (´ABT´) the control point.

Note: After a type-in of ´NEW´ requests stored on a possibly existing request-file ´lfn´ are lost. So be sure !!

-- lfn SIZE ERROR __
N.CFO NEW OR ABT

An ´n.CFO´ type-in is required to either generate a ´NEW´ file ´lfn´ or to abort (´ABT´) the control point.

Note: After a type-in of ´NEW´ requests stored on a possibly existing request-file ´lfn´ are lost. So be sure !!

Two more errors are reported to the dayfile. In either case an analyst should be consulted.

-- ***** REQUEST ERROR FOR lfn *****
***** VERY SERIOUS - CALL ANALYST *****

-- ***** CATALOG ERROR FOR lfn *****
***** VERY SERIOUS - CALL ANALYST *****

5.1.1.6 Stage III Messages - Mainframe Logical ID Check

In a multi-mainframe environment a special logical ID (LID) has to exist for the mainframe running SMS. A flashing B-DISPLAY message

-- TYPIN -ADDID,___- REQUIRED.

reminds the central operator. The message will disappear after the provision of the LID.

(Note that this LID is automatically provided by the peripheral program lUI, the ECMWF ID handler).

5.1.1.7 Stage IV Messages - Restart Mode

If the ´RESTART´ control card parameter was used restart mode will be invoked as last action in the initialisation process. (The same function is also performed in response to the ´REST´ command).

If any jobs supposed to be active are not in the computer system the operator is informed by a flashing B-DISPLAY message:

-- family/jobname MISSING -- TO BE RESTARTED
N.CFO YES OR NO

or

-- family/jobname MISSING -- TO BE RESTARTED
N.CFO YES OR NO -- NORESTART BIT SET

to which he has to respond accordingly. Any job supposed to be active which is already in the CYBER output queue will cause a message to be sent to the terminal:

-- family/task: jobname ALREADY IN OUTPUT QUEUE
family/task: NO NOTIFICATION OF TERMINATION YET

and the following flashing message to the supervisor's B-DISPLAY:

-- family/jobname ALREADY IN OUTPUT QUEUE
WAITING FOR N.ACK

This message has to be acknowledged with an 'n.ACK' type-in to ensure operator notification as the terminal message might not get sent if the message terminal is down.

This situation should be carefully investigated. Under normal circumstances it should only occur if jobs have been terminated by deadstart recovery procedures (outputs of deadstart recoveries to be checked for that) in which case the jobs should be restarted using 'RUNTSK' and the outputs should be either evicted or diverted to central (the above sequence being important).

There is also a possibility that CRAY jobs and to a lesser extent CYBER jobs may become affected due the fact that communication from the CRAY is not a synchronous process and/or certain requests might still reside on the request file which has not yet been processed. In these cases the 'REST' command should be used a couple of minutes later which will show in most cases that the problem has been resolved. Any files still being reported should be treated as follows:

-- Divert output files to central printer and examine the dayfile.

-- If the jobs ran to completion, do not restart them; otherwise a RUNTSK is necessary.

If an output has been diverted to the central printer any subsequent restart will report this job as missing in the system (as explained above) as long as it has not been restarted.

5.1.2 ´n.X OPS,command.´ Interface (***)

All supervisor interaction commands (cf. next chapter) can be issued from the CYBER console:

```
┌──────────────────────┐
│                      │
│  n.X OPS,command.    │      (***)
│                      │
└──────────────────────┘
```

This interface to the supervisor is to be used once INTERCOM is not available.

At the present moment there is no direct message logging facility in case INTERCOM is not available. ´n.X OPS,LOGFILE,PRINT.´ can be used in this situation to print the logfile on the central line printer (cf. description of ´LOGFILE´).

5.2 SMS Operator Commands

Supervisor interaction commands are normally issued from the command terminal. Please note that the INTERCOM routing ID of that terminal (which can be found out by typing ASSETS) has to be the same as the one shown in the supervisor´s B-DISPLAY (cf. ´B-DISPLAY Description´) for the command terminal. Commands issued from other terminals are ignored by SMS.

5.2.1 ALLMES - Set/Clear ALLMES Indicator (***)

```
┌──────────────────┐
│                  │
│         │ ON     │
│ ALLMES,│        │      (***)
│         │ OFF    │
│                  │
└──────────────────┘
```

The ´ALLMES´ command is intended to set or clear the ALLMES indicator (cf. ´B-DISPLAY description´ of SMS), eg. to instruct the supervisor to send all messages to the message logging terminal (´ON´) or to send just those of general interest (´OFF´).

5.2.2 DATE - Initialise New Suite (***)

```
DATE,suite type,day,date {,time}        (***)
```

The ´DATE´ command is the only command to initialise a new suite.

suite type One letter describing the suite to be initialised.

For installation defined suites see description of ´SMS´.

day Day of the week for ´date´ (MON, TUE, WED, THU, FRI, SAT, SUN)

date Date for suite in the format: yy/mm/dd.

time Optional start time for the suite in the format: hh:mm.

A ´DATE´ command for a suite with active/incomplete families and/or inactive families which are stored in the Delay Table DT will produce a message:

```
..... INCOMPLETE FAMILIES -- GO/DROP
(-GO- FOR NEW DATE/-DROP- TO IGNORE)
```

´DROP´ should be typed if the request is to be ignored. ´GO´ will cause the supervisor to ´go ahead´ to initialise a new suite.

In case of the above message ´STATUS,suite type,INCOM´ can be used (after ´DROP´) to see which families are affected.

5.2.3 EXEC - Set E-bit(s) (***)

```
                  | ,family n {,F}
EXEC,family 1     |                    (***)
                  | ,,F
```

Set the EXECUTABLE-bit (E-bit) for ´family 1´ or the ordered interval [family 1,family n]. A third optional parameter ´F´ can be specified if families shall be re-executed for some reason. This will be only necessary in exceptional cases and should be treated very carefully.

Note that the E-bit has to be set for a family before anything can 'happen' to it, eg. before the supervisor will work with it.

5.2.4 EXST - Set E-bit(s) And Start First Family (***)

```
┌──────────────────────────────┐
│                 ┌──────────┐ │
│                 │,family n {,F}│
│ EXST,family 1   │          │ │      (***)
│                 │,,F       │ │
└─────────────────└──────────┘─┘
```

Set the EXECUTABLE-bit (E-bit) for 'family 1' or the ordered interval [family 1,family n] and start 'family 1'. A third optional parameter 'F' can be specified if families shall be re-executed for some reason. This will be only necessary in exceptional cases and should be treated very carefully.

Note that the 'EXST' command is an abbreviated and thus recommended form for:

```
EXEC,family 1,...
START,family 1
```

The effect of a family being started is that first of all the START-bit (S-bit) will be set for that family. Secondly the family will be activated unless there is a start time and/or an end time for the family involved which might result in the family being entered into the so-called Delay Table (DT) to be activated later on or it might result in just the family E-bit being set in case of an end time which has already been reached. Activating a family actually results in the ACTIVE-bit (A-bit) being set and in all tasks (jobs) without events in that family being initiated.

5.2.5 INITMES - Initialise Message Terminal (***)

```
┌──────────────────┐
│ INITMES {,tid}   │      (***)
└──────────────────┘
```

The 'INITMES' command is intended to initialise the message terminal. It is to be used in case the supervisor's B-DISPLAY message shows 'DOWN' in the message terminal ID field. This is the case if INTERCOM was down or the terminal was logged-out and the supervisor attempted to send a message to that terminal. Note that the command is only necessary if 'DOWN' is shown in the B-DISPLAY. 'tid' is for analyst use only to initialise a message terminal different from the command terminal.

5.2.6 LOGFILE - Logfile Handler (***)

```
|                                      |
|          | PRINT {,ALL} {,ALLMES}    |
| LOGFILE, |                           |        (***)
|          | ARCHIVE                   |
|______________________________________|
```

This program is used to handle the supervisor´s logfile.

´LOGFILE,PRINT´ will print those messages from the logfile on the central line printer which usually appear on the message logging terminal. Note that only messages since the last ´LOGFILE,PRINT´ are affected. Thus the outputs of subsequent calls form the complete operator´s log.

If the parameter ´ALLMES´ is added all logfile messages will be included in the output. This feature will normally used by analysts only.

The parameter ´ALL´ in the end will print messages from beginning of information of the logfile (and not since last position).

The output listings are marked as LOGFILE.

´LOGFILE,ARCHIVE.´ will re-catalog the logfile as the next highest cycle of ´LOGFILEARCHIVE´ and generate a new logfile. This feature shall be used in weekly operator procedures to avoid an indefinite growing of the file.

5.2.7 PROPQT - Process Output-Queue-Table (***)

```
|          |
| PROPQT   |        (***)
|__________|
```

Process Output-Queue-Table. This command should be used if the output of aborted jobs is required immediately (which usually would be printed later - cf. ´OPQ´-parameter for SMS).

This command is only meaningful in a situation after the ´task-abort´ message has been received at the message logging terminal.

5.2.8 REST - Restart Suite(s) (***)

```
REST {,xyz}
```

(***)

Invoke restart processing for the indicated suites 'xyz' (each character has to be a valid suite type). If no parameters are used the function is executed for all (installation) defined suites (for details see description of 'Phase IV -- Restart Overlay' for the supervisor).

5.2.9 RUNTSK - Run Task (***)

```
RUNTSK,family,task
```

(***)

The 'RUNTSK' command can be used to run a task (job) independent from the family concept. Both parameters family name and task name are mandatory.

Care is required since re-running tasks can have disastrous results.

5.2.10 SETIM - Set/Clear Time(s) For Family Or Task (***)

```
SETIM,FAMILY=family {,TASK=task} {,START=hh:mm}
      {,END=hh:mm} {,P=hh:mm}
```

(***)

SETIM can be used to set/clear start, end or periodic restart times for families or set/clear start times for particular tasks.

Any time has to be in the format 'hh:mm'. A value of '0' will clear any associated time.

Note that using the 'SETIM' command with 'START=0' is the only way to force the start of a family or a task which has an associated start time and is stored in the delay table DT.

5.2.11 SETUP - Set-up Command/Message Terminal (***)

```
+-------------------+
|                   |
|  SETUP            |      (***)
|                   |
+-------------------+
```

This procedure has been tailored to set-up the command/message terminal for ECMWF´s requirements. It should always be called after a ´LOGIN´.

5.2.12 SMSLOG - Procedure To Handle Back-up Logfiles (***)

```
+-----------+
|           |
|  SMSLOG   |      (***)
|           |
+-----------+
```

This procedure should be included into the set of weekly operator´s procedures. The idea being that the archived logfiles (cf. description of ´LOGFILE´) are all purged after a full permanent file dump followed by another ´LOGFILE,ARCHIVE´-call to archive the current logfile which will then be available for the following week for inspection.

5.2.13 START - Start Family Or Task (***)

```
+-------------------------------+
|                 | ,task {,F}  |
|  START,family   |             |      (***)
|                 | ,,F         |
|                               |
+-------------------------------+
```

Start a family or a task. A third optional parameter ´F´ can be specified if some families or tasks shall be ´re-started´ for some reason. This will only be necessary in exceptional cases.

The effect of a family being started is that first of all the START-bit (S-bit) will be set for that family. Secondly the family will be activated unless there is a start time and/or an end time for the family involved which might result in the family being entered into the Delay Table DT to be activated later on or it might just result in the family E-bit being set in case of an end time which has already been reached. Activating a family actually results in the ACTIVE-bit (A-bit) being set and in all tasks (jobs) without events in that family being initiated.

Starting a task results in the START-bit (S-bit) for the task being set . If there is no start time involved the task will be initiated and the task ACTIVE-bit (A-bit) will be set. In case of an associated start time a Delay Table entry will be made for that task.

Note that the commands ´START,family,task,F´ and ´RUNTSK,family,task´ are equivalent. For cautions see description of the ´RUNTSK´ command.

5.2.14 STATUS - Status Enquiry Facility (***)

```
                                  (***)
```

STATUS,	suite type
	suite type,TERM
	suite type,INCOM
	suite type,DT
	family {,task}

The ´STATUS´ command can be used to obtain status information about active suites.

The first two formats will print the complete status of a suite on the central line printer or at the command terminal. The banner page for a line printer output listing is ´STATUSx´ where ´x´ denotes the suite type. Please note that a terminal print out can be very time consuming!

The third format will give the complete status of all incomplete families and all inactive families stored in the Delay Table DT at the command terminal.

The fourth format will print the contents of the Delay Table DT for the indicated suite.

The last format can be used to obtain a status printout of a particular family or one task within that family. Please note the convention within the SMS-system that the first letter of a family name and the first two letters of a task name are always equivalent to the suite type. Thus the suite can uniquely be determined from for example the family name.

Note that everything is written to the file OUTPUT if ´STATUS´ is called from a non-INTERCOM job (eg. ´n.X OPS´ interface etc.) as otherwise no output could be obtained.

5.2.15 STOP - General Stop Command (***)

```
┌──────────────────┐
│                  │
│      | SMS       │
│ STOP,| MES       │      (***)
│      | suite type│
│                  │
└──────────────────┘
```

The 'STOP,SMS' command should be used to stop the supervisor. For completeness it should be noted that the supervisor's control point can also be dropped or killed at the console to achieve the same.

In the form 'STOP,MES' the supervisor can be instructed to logically 'terminate' the message terminal, eg. messages are no longer sent to the terminal ('INITMES' is the reverse command).

In the final format 'STOP,suite type' a suite can be terminated. **Note that this is the only way of removing a suite from the list of active ones.**

An attempt to stop a suite with active/incomplete families and/or inactive families which are stored in the Delay Table DT will produce a message:

```
..... INCOMPLETE FAMILIES --- GO/DROP
(-GO- TO STOP SUITE/-DROP- TO IGNORE)
```

'DROP' should be typed if the request is to be ignored. 'GO' will cause the supervisor to 'go ahead' to stop the suite.

In case of the above message 'STATUS,suite type,INCOM' can be used (after 'DROP') to see which families are affected.

5.2.16 SUITDAT - Date And START Time Of Active Suites (***)

```
┌──────────┐
│          │
│ SUITDAT  │      (***)
│          │
└──────────┘
```

Print date and start time for every active suite.

5.3 Other SMS Interaction Commands

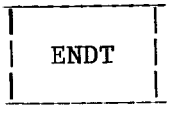
The following programs (apart from 'SMSFCT') can be used in the JCL of tasks/jobs initiated by the supervisor itself. The availability for CYBER or CRAY jobs is indicated for each of them.

5.3.1 ABTT - Task-Abort Notification

'ABTT' is used to send a 'task-abort' notification to the supervisor. The control card is preferably placed after an 'EXIT,S' statement in the control card stream (cf. also 'ENDT') of a job.

'ABTT' is available for CYBER and CRAY JCL.

5.3.2 ENDT - Task-End Notification



'ENDT' is used to send a 'task-end' notification to the supervisor. The control card is preferably placed before an 'EXIT,S' statement in the control card stream (cf. also 'ABTT') of a job.

'ENDT' is available for CYBER and CRAY JCL.

A typical job should be (NOS/BE JCL):

```
JOB,... .
ACCOUNT,... .
  ...
  ...
ENDT.
EXIT,S.
ABTT.
```

Please note that the above construction will only give the desired effect if the job is not killed by either the operator or the user itself in reprieve processing.

5.3.3 ISSEV - Issue Event

```
ISSEV,event
```

'ISSEV' can be used in a control card stream to issue an event to the supervisor. 'event' has to be a 3-digit (!) event number. This routine has been designed to be compatible with the procedure interface 'SETEV (event)'.

'ISSEV' can only be called within CYBER JCL.

5.3.4 ISSWMES - Write Message To Terminal

```
ISSWMES,mes
```

'ISSWMES' allows to send a message to the message terminal. 'mes' is the message to be sent which should preferably be terminated by the escape character '^'. The routine has been designed to be compatible with the procedure call 'WMES (message)'.

'ISSWMES' can only be called within CYBER JCL.

5.3.5 SMSFCT - External Calling Facility

```
SMSFCT,FCT=...,JN=...,TID=...,SUITE=...
  FAMILY=...,TASK=... {,EVENT=...} {,F}
```

This program has been written to allow communication with the supervisor from other mainframes, eg. all relevant information can be passed on the control card.

Please note that calls to 'SMSFCT' are generated by programs (which have been initiated by the supervisor) which, of course, can compute all required parameters.

FCT — Function to be performed by the supervisor:

ABTT: 'Task-abort' notification.
ENDT: 'Task-end' notification.
SETEV: Issue event (requires EVENT=...).
STATA: Start task.
STAFA: Start family.
WMES: Write message to message terminal (the message has to be the first line on the next partition on the INPUT file).

JN — Jobname of calling job (7 characters).

TID — Remote batch ID of calling job.

SUITE — 1-character suite type.

FAMILY — Family name of calling job.

TASK — Task name of calling job.

EVENT — Event to be issued (5 digits).

F — 'F' parameter for STATA or STAFA. For details see description of 'START' and 'RUNTSK'.

5.4 Special Programs

5.4.1 ATTARLF - Attach Archived Logfile

```
┌──────────┐
│ ATTARLF  │
└──────────┘
```

This call should be used to attach the highest cycle of the archived logfile(s) within back-up procedures (cf. description of 'LOGFILE').

5.4.2 INGEN - Initialisation File Handler

```
┌──────────────────────────────────────────────┐
│ INGEN,TIDCT=...,TID=...,CYBER=...,CRAY=...   │
│   CY1=...,CY2=...,CR1=...,CR2=...            │
└──────────────────────────────────────────────┘
```

Set-up, display, change supervisor's initialisation file. 'INGEN' has to be run before supervisor operation can be started. The intention is to define default terminal IDs and more important logical mainframe IDs used by the CRAY station(s) and/or the GEMINIs or APOLLOs (ECMWF version of GEMINI) to transfer input files. These mainframe IDs are used by the supervisor as ST-parameters on jobcards for its own generated jobs.

Under normal circumstances the file need not to be changed but operators might have occasionally to do so.

TIDCT Command terminal ID.

TID Remote batch ID(s) for permitted users.

CYBER LID for both CYBER(s).

CRAY LID for both CRAY(s).

CY1 LID for CYBER1.

CY2 LID for CYBER2.

CR1 LID for CRAY1.

CR2 LID for CRAY2.

NOTES:

1. The 'TIDCT' parameter is mandatory and has to be a 2-character combination which, of course, should be the same as the INTERCOM ID for the command terminal.

2. The 'TID' parameter is optional. Up to 4 TIDs, separated by slashes ('/'), can be specified. 'TID=0' will clear the field of TIDs.

   TIDs recorded on the initialisation file are displayed in the supervisor's B-DISPLAY.

3. Mainframe IDs:

   CYBER and CRAY LIDs have to exist on the initialisation file.

   An ID of 'ANY' for CYBER, CY1 or CY2 will result in a blank 'ST' parameter on the generated job card, thus saving FNT space.

   'CYBER=ANY' should always be used.

   If CY1, CY2, CR1, CR2 are not specified they will be defaulted to CYBER and CRAY respectively.

4. Calling 'INGEN' without parameters will list the contents of the initialisation file.

5. If changes have been made using INGEN the supervisor has to be dropped and called again to read the updated information from the file.

5.4.3 LOGANAL - Logfile Analyser

```
LOGANAL,SUITE=.,DATE=yy/mm/dd {,FULL} {,I=lfn} {,L=lfn}
```

SUITE — Mandatory 1-character suite type for suite to be analysed.

DATE — Mandatory date to be analysed.

FULL — Analyse full logfile, eg. from BOI. In the default case analysis will be from the last ´position´ (cf. description of the ´POSLF´ parameter for the ´SMS´ call and description of the ´LOGFILE´ command).

I — Optional input filename to be analysed.
Keyword: LOGFILE.
Default: No local file.

L — Optional output filename.
Keyword: OUTPUT.
Default: No local file but central line printer output. (Banner page: ´LOGANAL´)

5.4.4 TTGEN - Task Table Generator

To establish the necessary work files a so-called task table generator has to be used having the definition of a suite as input (the format is described in a different chapter).

```
TTGEN,suite type {,IF=...} {,LF=...} {,DF=...}
  {,PF=...} {,ID=...}
```

suite type Mandatory 1-character suite type (note that suite types are installation defined, cf. description of ´SMS´).

IF Input file -

Default: INPUT.
Keyword: INPUT.

LF Output file -

Default: OUTPUT.
Keyword: OUTPUT.
LF=0: no output to be produced.

DF Deck file -

Default: 0 - no deck file supplied.
Keyword: COMPILE.

PF Catalog generated task table file -

Default: no catalog.
Keyword: catalog file with default setting.
PF=pfn: use ´pfn´ and default ID.

ID Permanent file ID (only if ´PF´ specified) -

Keyword: default Id.
ID=id: use ´id´.

NOTES:

1. A task table file is only generated in an error-free case.

2. A final work file has always to include the deck file (´DF´ parameter). If the job decks are present a flag is set in the file which is checked during ´DATE´ command processing.

3. Every task defined in the suite has obviously to have a corresponding job in the deck file.

4. If the ´PF´ parameter is used (and not ID), eg. the generated file is going to be catalogued as the default file also to be used by the supervisor, it should be noted that the file is only catalogued if no corresponding back-up file exists. This is due to the fact that both files are necessary for supervisor operation but contain conflicting data. The suite should be stopped using ´STOP,suite type´ (the only way to get rid of a back-up file) followed by a call to ´TTGEN´.

5. When cataloging new task table files be aware that always new cycles will be generated. As NOS/BE only allows 5 cycles the old versions should be purged from time to time:

```
PURGE STAS_,ID=SMS,PW=SMSPWSMS,LC=1.
```

(For correct pfn, id and pw check COMPTXT definitions).

6 The Technical Implementation Of The SMS Subsystem

6.1 Preliminaries

The Supervisor-Monitor-Scheduler (SMS) system has been designed and implemented with the following requirements:

-- Implementation As Close As Possible To Standard O/S Software

This fact has been considered very important to not jeopardize future operating system upgrades which is essential for the plans of a centre like ECMWF.

-- Modularity

SMS to be written in a modular fashion with modern software design technologies employed.

-- Maintainability

The SMS Subsystem, as one of the central pieces of software at ECMWF, had to be implemented such that the maintenance and support function could be carried out by different staff.

-- Parametrization

A highly parametrized subsystem was to be developed aiding future changes.

-- Efficiency

As SMS would be continuously running it should not impose extraordinary overhead on to the operating system and other subsystems.

-- Flexible Operator Control

The computer operator shall be given proper tools to control the running of SMS and all the suites.

-- Security

In the SMS Subsystem should also be incorporated a level of security at a similar level of security used in all other parts of the computer systems at ECMWF.

6.2 Programming Languages

The SMS Subsystem utilises the following programming languages:

SYMPL SYMPL is a CDC Systems Programming Language which has got similarities to PASCAL or PL/1; this statement should be understood in the way, that, if programs are written in SYMPL how they have to be written in PASCAL or PL/1 then similarities can be observed. The choice for SYMPL was made since it is a CDC product and there are language features like COMMON blocks and a flexible interface to the operating system level missing in PASCAL for example.

All major (CYBER) parts of the SMS Subsystem are written in SYMPL.

COMPASS COMPASS is the standard CDC Assembly Language.

It is being used for a small subroutine collection not easy to write in SYMPL and for presets of certain data structures. Furthermore, the SMS main overlay is written in COMPASS for efficiency reasons.

CFT CFT is the standard CRAY FORTRAN compiler.

All CRAY interface routines are implemented in CFT.

CAL CAL is the standard CRAY assembly language.

All CRAY operating system interface routines are implemented in CAL.

6.3 Symbol Definitions And System Texts

A major effort was devoted to avoiding the problems of large-scale software development utilising different programming languages, namely in the area of symbols being used by different compilers or assemblers. Within the SMS Subsystem all symbols are only defined once. A certain syntax had to be devised, of course, and, some people might say tricks, have been employed to derive the same symbol definition in each programming language.

6.3.1 Common Symbol Definitions

Those symbols used by more than one programming language are all contained in one of the three UPDATE common decks:

ASS-OPTS This common deck holds all general installation constants. It is accessable by COMPASS, CAL and SYMPL.

COMSYM This common deck contains all common symbol definitions for COMPASS and SYMPL.

CYBCRA1 CYBCRA1 contains all micros utilised by the CYBER/CRAY-communication interface. It is being used by COMPASS and CAL.

6.3.2 System Texts: COMPTXT, SYMDEFS, CALTEXT

Three main system texts have been created for the SMS Subsystem:

COMPTXT This COMPASS text contains all symbol definitions (including the definitions from the previous paragraph) and all special macros and micros.

COMPTXT basically defines all the names, eg. file name, overlay names, program names etc., being used by the SMS Subsystem. It also defines the overlay structure for SMS and the suites the SMS Subsystem is supposed to handle.

SYMDEFS SYMDEFS is the main SYMPL text which contains all symbols being used in the SYMPL assemblies. Furthermore, it contains useful syntax abbreviations to make written code more look like PASCAL. All data type definitions (STATUS lists) used by more than one SYMPL routine, all data extractor/compactor functions, the set operators and certain sets, all I/O subroutines, all routines for the 'file-request interface' and some other commonly used functions and subroutines have been defined as well. Please note, that these definitions are all SYMPL DEF statements, eg. they all expand to something more or less complicated and simply serve as to make written code more readable and reduce the error probability.

CALTEXT This text only contains macro definitions for CAL assemblies and the common deck ASS-OPTS for the assembly options. (The text does not contain other definitions because of the shortcomings of the CAL system text implementation).

6.3.3 Other SYMPL Texts

Given the symbol definitions in SYMDEFS many of the common data structures, used by the SYMPL subroutines, are defined as separate texts thus avoiding re-definitions or UPDATE common deck inclusions. Please note that SYMPL does not allow presets of variables in texts!

Special attention is drawn to texts SYMBITV (SYMPL definition of the bit vector), SYMLFNT (SYMPL definition of the local file name table for files STASx and STABx), SYMLFN2 (SYMPL definition of the local file name table for files UCPFILE and SMSFILE used for the file request interface) and SYMOVCT (SYMPL definition of the overlay control table used within SMS). All these texts utilise a special syntax and call UPDATE common decks which are also used within COMPASS assemblies.

6.4 The I/O Interface Used By The SMS Subsystem

As only the CYBER I/O routines are of any interest - since SMS is running on a CYBER - the CRAY CFT I/O interface will not be discussed.

6.4.1 Sequential Files

Sequential file operations are all performed using the I/O macros from CPUTEXT, eg. the interface to CIO=. Highlevel interface routines (similar to routines in the FASTIO package) do exist for all required operations.

6.4.2 Random Files

For performance reasons most of the I/O carried out by SMS is random access, eg. random addresses are always on PRU (Physical Record Unit) boundaries.

The interface provided is a very neat little routine called PIO - Physical I/O, with entry points PTPC (Put PRUs) and GTPC (Get PRUs) both with an extensive SYMPL parameter list. Because of this parameter list, and because of the possibility for misspellings, a large set of SYMPL special case interface routines has been defined using DEFs in text SYMDEFS. These routines are typically defined like READ$xx, WRITE$xx, REWRITE$xx etc. where xx describes some sort of a buffer like FI (Family Index), DT (Delay Table) etc. as their positions on the various files are known; parameters to be provided for these calls are only a local file name and a recall indicator.

6.5 The SMS Logfile And The Message Logging Interface

SMS has been designed to log all actions it is performing. This so-called logfile is a permanent file and is fully maintained (created, extended, positioned) by SMS. Messages can be designating to go the logfile only or to the logfile and the so-called message logging terminal for the SMS operator. The latter interface has been implemented based on the standard INTERCOM-user to INTERCOM-user communication interface (MES). The peripheral program MES has been modified to allow any job to send messages to a logged-in INTERCOM terminal. Note that MES uses the OUTSUB write routines and not 3TT. Using OUTSUB implies that only a limited number of INTERCOM small buffers will be allocated for each user table with the consequence that on subsequent calls all the MESs will go into the delay stack eventually causing PP saturation once asynchronous terminals are used and the terminal is waiting in sending mode (bar pressed etc.). To avoid this the actual SMS subroutine TERMESS will generate a flashing B-Display message for the operator (´** sending message **´) prior to the call to MES which is cleared after completion (note that MES is called in auto-recall).

6.6 Notes On The Implementation Of SMS

6.6.1 System Control Point

SMS has been implemented as a System Control Point (SCP) . The NOS/BE System Programmer´s Reference Manual (SPRM) should be consulted for a detailed description of the SCP features.

The SCP is required for the SMS-Operator-Communication. It is also used for other requests from jobs/tasks running on the same mainframe as SMS. The SMS main overlay should be understood for SCP processing as well as the routines SFENDT and CALLSSC in conjunction with the parameter block /UCPBUF/.

6.6.2 File-Request Interface

As explained earlier the SCP interface can only be utilised on one mainframe. For the job-to-SMS communication for those jobs running on a different mainframe the so-called file-request interface is provided. Two files SMSFILE and UCPFILE are being used for this purpose. Both are permanently attached by SMS (when running): SMSFILE with RW=1, UCPFILE with MR=1. Both files will be attached by the requesting jobs with the permission requests reversed. The first PRUs of each of the files contain equal length bitmaps whereby bit_n designates the nth PRU on UCPFILE. These bits are used by the requesting jobs to determine the next 'free' PRU on UCPFILE where they will then write their request followed by a toggle of the very bit in the bitmap on UCPFILE as well (note that only one user job can have the two files attached at any one time). SMS also uses the bitmaps to determine whether there are new requests on UCPFILE. If so, the first request is read from the file followed by a change of the bit in the SMSFILE bitmap and processing of the request continues as for any other SCP request.

6.6.3 The CYBER User Interface

In order to be flexible and to reduce memory requirements the CYBER User-Job-to-SMS interface has been implemented using CAPSULES (cf. NOS/BE LOADER Reference Manual). At the time of implementation no good description existed of how to write and handle CAPSULES: deck 'SMSFDL=' within the SMS UPDATE program library should be consulted (and the CYBER installation job as well) to understand the principle.

Important to note is that CAPSULE loading has been provided from non-incapsulated program units and from already loaded CAPSULES as well (without using any stack or other fancy machanism: use has simply been made of the table generated by the ENTHDR macro cf. UPDATE common deck: FDLENTTAB).

All CYBER user functions will first load a CAPSULE which will attempt to send a System Control Point request. If the SMS subsystem is not initiated another CAPSULE will be loaded to service the file-request interface. Thus minimal overheads are imposed on the user jobs.

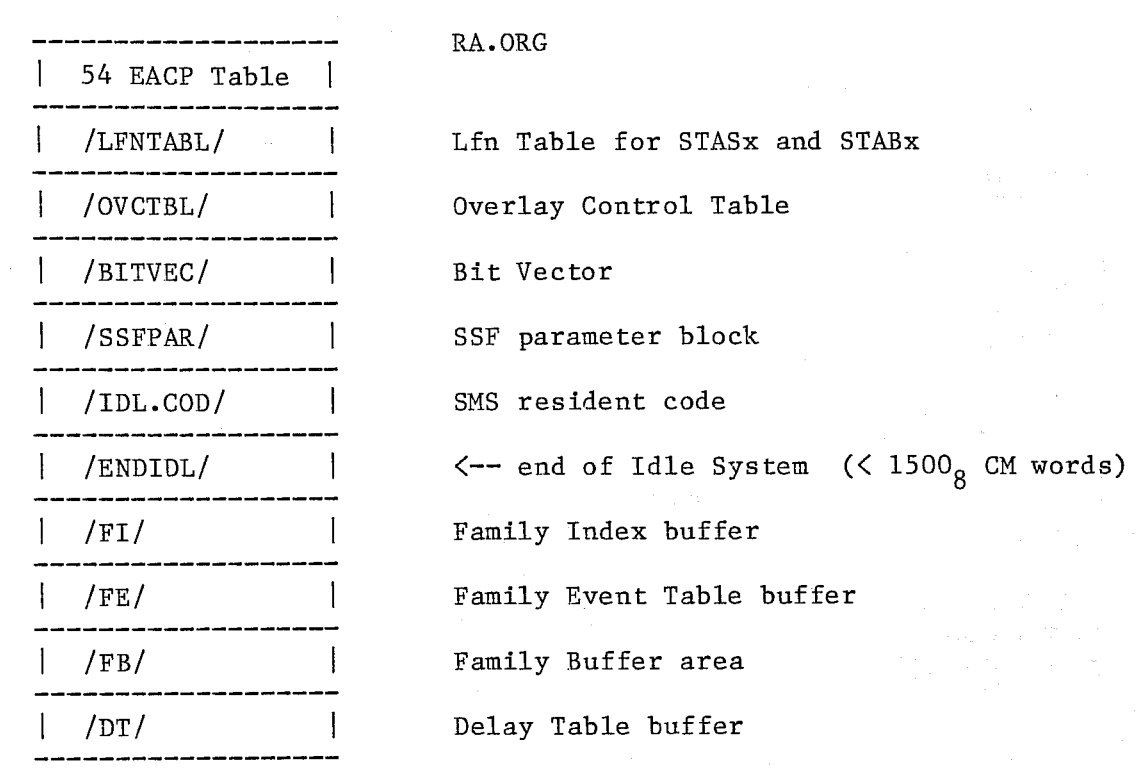
6.6.4 Overlay System

SMS has been written using an overlay structure. All overlay names and the overlay structure are defined in text COMPTXT. The same structure is also preserved in the so-called overlay control table (note that this is the only structure which is defined twice; but if changes are made in one definition and not in the other the assembly will abort with compilation errors).

One of the objectives was to write a main overlay with minimum field length requirements. The main overlay, on the other hand, should also be able to handle most of the initial request processing like validation, request decoding etc. and it should be able to periodically call-up all the necessary maintenance functions.

As mentioned earlier the SMS main overlay is written in COMPASS. Because of that it was possible to assemble everything, eg. the resident part of SMS during execution and all other buffers required in common blocks (implying that the zero block length is 0!). This approach helped to overcome the problems associated with the relocation of code and common blocks by the CYBER LOADER. It also helped to define all other overlays in a 'normal' way, eg. no fancy tricks are employed like overlay skeletons etc.

The following diagram gives an illustration:



It is easy to see from the above that all buffer areas are accessable from higher level overlays (via common blocks) and that SMS can also perform easy memory management as the end of the idle system is well defined.

6.6.5 Reprieve Processing

Some effort has been devoted to properly process an operator drop or operator kill of the SMS control point. It is obvious that any user request must be completed and that SMS has also to ensure that NOS/BE is not sending any new requests after a drop/kill. Routine PEX (within the SMS idle system), which is the error exit after reprieve, should be carefully read to understand the principle.

An operator drop is functionally equivalent to the 'STOP,SMS.' command with the SMS dayfile to be printed on the central line printer while an operator kill will discard any output.

All other error codes will cause a dump of SMS' current field length followed by an abort function irrespective of any functions in progress or any outstanding requests.

6.6.6 CRAY Q-Enqiry Facility

For the 'RESTART' function the CRAY Q-Status has to be obtained. As the CRAY link protocol and also the CRAY station support this function the routine CRAQST is called which will always be loaded from SYSLIB (ECMWF feature). SMS will have to be re-assembled once CRAQST changes; routine PREST, the 'RESTART' processor, will have to be changed once CRAQST output formats change.

6.6.7 Internal Documentation Of SMS

This paragraph describes the internal documentation extracted from the SMS source text in a structured way.

6.6.7.1 Initialisation Code

```
--  ISSUE VERSION MESSAGE
--  GENERATE B-DISPLAY MESSAGE FOR THE CENTRAL OPERATOR
--  CHECK JOB ORIGIN
--  ENSURE 5400 LOADER TABLE (EACP)
--  CHECK FOR CEJ HARDWARE
--  SAVE JOBNAME IN *JN*
--  SET CURRENT FIELD LENGTH
--  COMPLETE LDV PARAMETER BLOCK
--  WRITE ACCOUNT CODE INTO CONTROL POINT AREA (W.CPFACT)
--  TRY TO ATTACH "LOGFILE" FROM THE SYSTEM DEFAULT FILE BASE.
    IF THE FILE EXISTS (SUCCESSFUL ATTACH) WE WILL CHECK FOR
    READ, MODIFY AND EXTEND PERMISSION ON IT AND SKIP TO EOI IF
    SO. IF IT DOES NOT EXIST WE REQUEST A NEW FILE AND CATALOG
    IT. IF IT EXISTS BUT IS ATTACHED WE CAN ONLY INFORM THE
    OPERATOR AND ABORT. (THIS IS USEFUL IN CASE OF TWO CYBER
    MAINFRAMES TO ENSURE ONLY ONE COPY OF "PGM.SMS" IS
    RUNNING IN THE LINKED ENVIRONMENT (SRMS)).
--  CHECK PERMISSIONS ON LOGFILE
--  ATTACH "INITLFN"
--  READ 1ST PRU FROM "INITLFN"  -  USE /FE/ AS BUFFER
    RETURN "INITLFN"
--  COPY TID FOR COMMAND TERMINAL FROM *TIDS* INTO *MES$PAR* AS
    DEFAULT MESSAGE TERMINAL IDENTIFIER AND SET *S$MESS* TO
    INDICATE AVAILABILITY OF THE TERMINAL (CF. *MESS*).
--  PROCESS ARGUMENT LIST SUPPLIED IN CALL.
--  'POSITION' LOGFILE TO CURRENT EOI IF 'POSLF' PARAMETER PRESENT
--  INITIALISE LOW CORE VARIABLES
--  SET REPRIEVE MASK
--  WRITE DATE - VERSION MESSAGE TO LOGFILE AND TERMINAL
    AND THE CONTROL CARD COPY TO THE LOGFILE
--  COMPLETE B-DISPLAY MESSAGE BUFFER AND ISSUE MESSAGE
--  LOAD -INIT- OVERLAY
--  ISSUE MESSAGE FOR ERROR ENCOUNTERED DURING INITIALISATION
    AND ABORT PROGRAM WITHOUT DUMPING FL.
--  KEYS  -  ASSEMBLE CONTROL CARD SKELETON HERE
--  PROCESS ARGUMENT LIST SUPPLIED IN CALL.

    PLEASE NOTE THAT ARGUMENTS WILL BE PROCESSED USING *ARG=*
    FROM *COMFARG* (NOT *COMCARG* !), I.E. PARAMETERS WILL BE
    EXTRACTED FROM *RA.CCD+...*. (*RA.ARG+...* HAS ALREADY BEEN
    OVERWRITTEN BY PREVIOUS INITIALISATION CODE).
--  CHECK 'RESTART=...' PARAMETER(S)
--  LD$REST  -

    RETURN FROM *OV$INIT* ASSEMBLED IN /DT/ TO LOAD *OV$REST*
--  *REST$* TO BE DEFINED IN THIS PART. IT WILL BE PICKED UP
    BY OVERLAY *OV$INIT* TO BE PASSED TO *OV$REST* BEFORE IT
    GETS BASHED.
--  THE LOCATION COUNTER FOR THE FINAL 'END' STATEMENT MUST
    BE ZERO. EXECUTION WILL FAIL OTHERWISE.
```

6.6.7.2 Internal Documentation Of SMS´ IDLE SYSTEM

```
--  I D L C    -    I D L

    MAIN IDLE LOOP
--  KILL THIS PROGRAM IF *S$KILL* IS SET
--  TERMINATE THIS PROGRAM IF *S$DROP* IS SET
--  CHECK FOR NEW UCP REQUEST (ALSO ENTERED FROM *TERMIN*)
--  NO UCP REQUEST IN *RA.SSC*

    IF BIT *S$MORFR* IS SET (I.E. MORE REQUESTS ARE ON THE REQUEST
    FILE) WE WILL LOAD OVERLAY *OV$FILE* TO HANDLE THAT CASE.
--  NO UCP REQUEST IN *RA.SSC* OR ON FILE
--  OUTPUT QUEUE TABLE PROCESSING
--  PERFORM MISCELLANEOUS TASKS PERIODICALLY WHILE IDLE
--  PROCESS DELAY TABLE
--  HANDLE FILE-REQUESTS
--  INSERT HERE MORE TASKS TO BE PERFORMED
--  GO INTO PERIODIC RECALL AND IDLE DOWN AFTER *TI.IDL* EXPIRED
--  UCPREQ  -

    THIS PART TO BE ENTERED AFTER A NEW UCP REQUEST HAS
    BEEN RECEIVED FROM EITHER NOS/BE (*RA.SSC* NEGATIVE)
    OR IF FILE-REQUESTS HAVE TO BE HONOURED (IN THIS CASE
    DIRECTLY ENTERED FROM THE OVERLAY)
--  FLAG IDLE DOWN TIMER TO BE SET; NO LONGER IDLE;
    REQUEST NOT FROM COMMAND TERMINAL
--  CHECK STATUS FIELD IN *AP*
--  STATUS IN *AP* IS EITHER 1 OR 2, I.E. WE HAVE TO ISSUE
    AN ´SF.ENDT´ FUNCTION FOR THIS UCP WITH ´UCPA´ SET TO -1
    TO CLEAR ALL WAIT-RESPONSE COUNTS AND LONGTERM CONNECTS
    ETC. (CF. *SFENDTC*).
--  PLEASE NOTE X6 = (AP) <> 0  FOR *FLAG* IN *SFENDTC*
--  CHECK UCP REQUEST TO BE FROM A CONSOLE INITIATED JOB
    OR FROM THE COMMAND TERMINAL

    WE FIRST CHECK IF THE JOB ORDINAL IN *AP+1* IS <= 17B IN
    WHICH CASE ALL CHECKS ARE SKIPPED (NOTE: *S$UCPCOM* IS
    ALSO SET TO ALLOW COMMANDS TO BE ENTERED FROM THE CONSOLE).

    THE CALLER-S TID IN *UCPBUF* WILL OTHERWISE BE COMPARED
    WITH THE LAST TWO CHARACTERS OF THE JOB NAME WHICH IS
    GENERATED FROM THE TID FOR INTERCOM CP-S (OLD FORMAT:
    IDIDIID, NEW FORMAT FROM LEV. 530 ONWARDS: IOIDOID).
    IF THEY MATCH THE TID IS COMPARED WITH THE TID FOR
    COMMAND TERMINAL AS RECORDED ON "INITLFN".
--  UCP IS NOT THE COMMAND TERMINAL

    ENSURE SUITE IS ACTIVE - TAKE SUITE TYPE FROM *JN*
--  ERROR: UCP REQUEST FOR SUITE WHICH IS NOT YET ACTIVATED
--  CHECK THAT REQUEST IS FROM A PERMISSIBLE JOB, I.E.
    THE TID IN THE UCP BUFFER HAS TO MATCH ONE OF THE
    TID-S IN *TIDS* OR HAS TO BE "TID", I.E. INITIATED
    BY "PGM.SMS" BEFORE.
--  ERROR: REQUEST FROM UNAUTHORIZED UCP
--  ENSURE THAT THE REQUESTED FUNCTION EXISTS
--  ERROR: NON-EXISTING OR NOT-USER-CALLABLE FUNCTION REQUESTED
--  ENSURE OVERLAY IS USER CALLABLE
--  SAVE POINTER TO *OVCTBL*, REQUEST ENOUGH FL, RESET
```

```
    *RA.SSC* UNLESS *OV.SYSFC* IS TO BE LOADED IN WHICH
    CASE A STOP-FUNCTION MIGHT HAVE BEEN ISSUED AND
    WE DON-T WANT TO RECEIVE MORE REQUESTS.
--  CHECK THE ´INDEX REQUIRED FLAG´ WHETHER AN INDEX IS TO BE
    LOADED OR NOT. IF SO WE HAVE TO BUILD CORRECT LFN-S IN
    /LFNTABL/
--  COMMAND TERMINAL MUST NOT REQUEST AN INDEX AS WE CAN-T EXTRACT
    THE SUITE TYPE FROM THE *JN*. INDEX LOAD WILL BE IGNORED.
--  DO WE HAVE TO LOAD THE INDEX ?
--  HERE AFTER INDEX LOAD HAS BEEN DELT WITH.
--  HERE AFTER ERRORS HAVE BEEN DETECTED FOR A UCP REQUEST

    ENTRY: X1 = (MESSAGE)    TO BE WRITTEN TO THE LOGFILE
           *SCR1* SET = 0 OR <> 0 FOR *SFENDTC*
--  IF BIT *S$FILREQ* IS SET (I.E. THE CURRENT REQUEST HAS BEEN
    LOADED FROM THE REQUEST-FILE) WE HAVE JUST TO CLEAR IT AND
    RETURN TO THE IDLE LOOP.
--  IN AN ERROR CASE WE WILL CALL *SFENDTC* FIRST BEFORE
    RESETTING *RA.SSC*. THIS IS NECESSARY SINCE A MEMORY
    DEADLOCK SITUATION COULD OCCUR IF *SFENDTC* WAS UNABLE
    TO HANDLE THE REQUEST AND WE HAD TO LOAD *OV$SFEND*
    ACCOMPANIED BY A POSSIBLE MEMORY REQUEST.
```

6.6.7.3 Subroutines Within SMS Main Overlay

```
--  BDISPC (MESSAGE)  -

    ISSUE B-DISPLAY MESSAGE FOR THE CENTRAL OPERATOR.
    JUST THE FIRST LINE <= 40 CHARACTERS WILL BE CONSIDERED.

    ENTRY: X1 = (MESSAGE)

    *MESSAGE* HAS TO BE IN C-FORMAT OR CAN BE EXACTLY 40
    CHARACTERS LONG.

    EXIT: B1 = 1

    USES: A - 6,1,0
          X - 6,5,1,0
          B - 3,1

    CALLS: BDIS2ND

--  BDIS2ND  -

    GENERATE 2ND LINE OF B-DISPLAY MESSAGE

    2ND LINE TO CONTAIN THE FOLLOWING:

    TID OF SYSTEM - "TID"
    TID FOR COMMAND TERMINAL
    TID FOR MESSAGE TERMINAL
    ´ALLMES´ FLAG - ´-A´
    ALL PERMISSIBLE TID-S AS FROM "INITLFN"
    ´DOWN´ IF NO MESSAGE TERMINAL
    ACTIVE SUITES AND LAST ACTIVE SUITE IF INDEX STILL LOADED

    EX.:
```

```
90/G1/G9   AWK1     OEF      X
90/Z9/45            EF
90/F4/DOWN AWK1     TFC      X

WHERE X DENOTES THE 'LAST ACTIVE SUITE' IF THE INDEX
OF THAT SUITE IS STILL LOADED.

USES: A - 7,6,1
      X - 7,6,4,1,0
      B - 1

MACROS: IFBIT, MESSAGE

--  FIRST WORD

--  SECOND WORD

--  THIRD WORD

--  FOURTH WORD

--  DMPFLC  -

DUMP ALL FIELD LENGTH

THIS ROUTINE IS WRITTEN WITHOUT ANY RJ-INSTRUCTIONS
AS IT IS ALSO CALLED BY *PEX*.

USES: A - 6,1
      X - 6,1

--  GENMESS  -

GENERATE COMPLETE MESSAGE IN *M.BUF*

1. PRODUCE TIME STAMP
2. MASK OUT ORIGIN IN *M.BUF+1,2*
3. COPY MESSAGE UP TO 12D WORDS IN C-FORMAT UNLESS TERMINATED
   BEFORE BY ESCAPE CHARACTER "ESC.CH".

ENTRY: B1 = 1
       B5 = (1ST WORD OF MESSAGE)
       X5 = WORD TO BE SUBSTITUTED IN C-FORMAT (<= 10 CHARS)
            IF "SUBST.CH" IS CONTAINED IN THE MESSAGE
       *M.BUF+1,2* PROPERLY PRESET (18 CHARACTERS) WITH ORIGIN

EXIT: B1 UNCHANGED
      B5 DESTROYED
      MESSAGE IN C-FORMAT WITH TIME STAMP GENERATED IN *M.BUF*

USES: A - 7,3,2,1
      X - 7,6,5,4,3,2,1,0
      B - 6,5,4,3,2

MACRO: CLOCK

THIS ROUTINE HAS BEEN PARTLY ADOPTED FROM *COMCSNM* WHICH
SOME PEOPLE MIGHT REMEMBER FROM THE GOOD OLD TIMES.
IT IS STILL GOOD TODAY.  (F.H.).

--  GETFL  -
```

```
    GET ENOUGH FIELD LENGTH FOR SUBSEQUENT LOAD

    ENTRY: X6 = POINTER TO *OVCTBL*
           B1 = 1

    EXIT: B7 = 0 OR 1  IF FIRST OR SUBSEQUENT LOAD OF OVERLAY

    USES: A - 4,2
          X - 6,4,2
          B - 7

    CALLS: MEM

--  KILL  -

    KILL THIS JOB

    USES: A - NONE
          X - NONE
          B - NONE

    MACROES: LOGMESS, SYSTEM

    CALLS: SYSEND

--  NEXT CARD IS *CALL CDLDBINDC

--  LDBINDC  -

    LOAD INDEX FROM BACK-UP FILE *STAB..* (NO RECALL).

    IT IS ASSUMED ON ENTRY OF THIS ROUTINE THAT
    *STAB..* EXISTS AND THAT ENOUGH FL IS AVAILABLE.

    THIS ROUTINES SHOULD ALWAYS BE CALLED IF THE INDEX IS
    TO BE LOADED FROM THE BACK-UP FILES *STAB..*.

    THE 'LAST ACTIVE SUITE' IN *SUITES* IS UPDATED AND BIT
    *S$LOADIN* CLEARED.

    USES: A - 6,5,2,1
          X - 6,5,4,3,2,1,0
          B - 1

    MACRO: CLEARBIT

    CALLS: PIO

--  MEM  -

    REQUEST CENTRAL MEMORY.

    ENTRY: X6 = NUMBER OF CM WORDS
           B1 = 1

    USES: A - 6,1
          X - 6,1

    MACRO: SYSTEM

    IF SYMBOL *MEM$MESS* IS DEFINED
```

PRODUCE LOGFILE MESSAGES FOR THE ACTUAL MEMORY REQUESTED.

USES: X - 5

MACROES: LOGMESS, SYSTEM

CALLS: COD=

-- MESS -

ISSUE INTERNAL MESSAGES.

ENTRY: X1 = (1ST WORD OF MESSAGE)
X2 = (1ST WORD OF MESSAGE ORIGIN - 2 WORDS LONG)
X5 = SUBSTITUTION WORD IF ANY IN C-FORMAT
(CF. *GENMESS*)
B7 = 0 MESSAGE TO LOGFILE ONLY
<> 0 MESSAGE TO LOGFILE AND MESSAGE TERMINAL

MESSAGES TO LOGFILE ONLY WILL HAVE A ´ ´ AS THE 10TH CHARACTER
OF THE FIRST WORD OF THE MESSAGE (WHICH IS THE TIME STAMP) -
MESSAGES TO LOGFILE AND TERMINAL WILL HAVE A ´.´ IN THAT POS.

MESSAGES ISSUED BY THE SUPERVISOR (AND NOT BY THE ´WRITE
MESSAGES´ FUNCTION) WILL HAVE THE SUPERVISOR-S JOBNAME
INCLUDED IN THE MESSAGE ORIGIN.

EXIT: B1 = 1

USES: A - 6,2,1
X - 6,2,1,0
B - 5

MACROS: WRITEC, IFBIT

CALLS: GENMESS, TERMESS

IF BIT *S$ALLMES* IS SET ALL MESSAGES ARE SENT TO
THE MESSAGE TERMINAL

-- OVL -

OVERLAY LOADER

ENTRY: B1 = 1
P.OVCTB = POINTER TO *OVCTBL* PROPERLY SET

EXIT: B5 = EPT ADDRESS

USES: A - 7,6,5,4,2,1
X - 7,6,5,4,3,2,1,0
B - 6,5

MACROS: LOADREQ, IFBIT, CLEARBIT

OVL WILL FIRST CHECK IF THE OVERLAY IS ALREADY
LOADED IN WHICH CASE JUST THE OVERLAY CONTROL TABLE
IS UPDATED (PLUS 1 CALL). AN ORDINARY OVERLAY
LOAD SEQUENCE IS OTHERWISE INITIATED AND THE CONTROL
TABLE UPDATED ACCORDINGLY (PLUS 1 CALL AND 1 LOAD).

```
    THIS ROUTINE ASSUMES THAT ALL OVERLAYS ARE GENERATED
    USING THE *OVLHDR* MACRO AND ALL OVERLAY NAMES ARE
    GENERATED BY THE *OVERL* MACRO. ALL PRIMARY OVERLAYS
    ARE LOADED AT MINFL OF THE (0,0)-OVERLAY AND SECONDARY
    OVERLAYS ARE LOADED AT MINFL OF THE CORRESPONDING PRIMARY
    OVERLAY.

    ENTERING THIS ROUTINE REQUIRES ENOUGH FL TO BE AVAILABLE.

--  IS PRIMARY OVERLAY ALREADY LOADED ? (L1 [X1] = L1 [PADDR])
    L2 [X1] = 0  <=>  SECONDARY OVERLAY LOAD
    OVERLAY ALREADY LOADED ?  (L1_L2 [X1] = L1_L2 [PADDR])

--  LOAD OVERLAY NOW

      X1 = 48/0,12/L1_L2
      B6 = POINTER TO *OVCTBL* FOR THIS OVERLAY.

--  OVL.S1  -

    UPDATE *OVCTBL* AND CLEAR BIT *S$LOADOV* AS OVERLAY IS
    LOADED. CLEAR ALSO BIT *S$IDTIM* TO INDICATE THAT THE
    IDLE DOWN TIMER HAS TO BE SET AGAIN AND *S$IDLE* AS WELL
    AS WE ARE NO LONGER IDLE.

    ENTRY: B6 = (ENTRY IN *OVCTBL*)
           X0 = MASK OF WHAT TO BE UPDATED
           X3 = TO BE ARITHM. ADDED TO TABLE ENTRY

--  PEX  -

    X-PACKAGE ADDRESS AFTER REPRIEVE

    THE FOLLOWING ACTIONS ARE TAKEN:

    -  IF *RA.SSC* IS NOT BUSY THE HIGH ORDER BIT IN *RA.SSC*
       WILL BE SET TO PREVENT NOS/BE FROM SENDING NEW REQUESTS.
       PLEASE NOTE THAT SETTING THIS BIT IS PERFORMED IN ONE
       60-BIT WORD WHICH IS SAFE AS ALL UCP REQUESTS ARE PASSED
       BY CPMTR WHICH OF COURSE RUNS IN MONITOR MODE AND ANY
       EXCHANGE JUMP WILL LET US COMPLETE THE INSTRUCTIONS IN
       THAT WORD (QED).

       *S$NOREQ* WILL BE SET IF WE COULD SET *RA.SSC* COMPLETE
       (THE BIT IS NOT SET BY DEFAULT).

    -  IF BIT *S$KILL* IS SET THE PROGRAM WILL BE KILLED.

    -  SET BITS *S$DROP* AND *S$PEX*.

    -  DUMP ALL FIELD LENGTH UNLESS OPERATOR KILL.

    -  IF ERROR CODES INDICATE OPERATOR DROP, KILL OR RERUN
       THE CORRESPONDING BITS *S$...* ARE SET AND *XJR* IS
       CALLED TO RESTORE THE X-PACKAGE AND RESUME EXECUTION.
       THE BITS *S$...* ARE CHECKED IN *TERMIN* TO PRODUCE
       LOGFILE MESSAGES.

    -  ALL OTHER ERROR CODES CAUSE AN ERROR MESSAGE TO BE
       PRINTED IN THE LOGFILE FOLLOWED BY A CPU ABORT.
    DO NOT USE ANY CODE MODIFYING INSTRUCTIONS UP TO
```

```
    THE POSSIBLE EXCHANGE JUMP FOR *XJR* (NOTE: ALSO
    *RJ* INSTRUCTIONS DO MODIFY CODE !!)

--  THE FOLLOWING EXCHANGE JUMP WILL CAUSE EXECUTION TO
    RESUME AT THE P-ADDRESS STORED IN THE X-PACKAGE.

--  NOW WE WILL TERMINATE THIS PROGRAM IRRESPECTIVELY OF ANY
    OUTSTANDING REQUESTS.

--  GTPC, PTPC, PIO  -

    PRU I/O ROUTINES

--  NEXT CARD IS *CALL PIO

--  GTPC  - GET PRU-S.

    CALLED AS: GTPC (FET[0],LFN,BUF,PRU NO.,NO. OF PRU-S,RECALL)

    USES: X - 6

    CALLS: PAR, PIO

--  PTPC  - PUT PRU-S.

    CALLED AS: PTPC (FET[0],LFN,BUF,PRU NO.,NO. OF PRU-S,RECALL)

    PRU NO. =  0   =>   WRITE AT EOI AND RETURN PRU ADDRESS
    PRU NO. <> 0   =>   REWRITE FROM INDICATED PRU

    USES: X - 6,3

    CALLS: PAR, PIO

--  PIO   - MAKE PRU I/O REQUEST.

    ENTRY: B1 = 1
           B2 = 1/0  FOR RECALL/NORECALL
           X2 = (BUFFER)
           X3 = PRU NO. / (RMS RETURN INFO)
           X4 = NUMBER OF PRU-S
           X5 = LFN
           X6 = FUNCTION CODE
           A1 = (FET [0])

    USES: A - 7,6,5,4,2,1
          X - 7,6,5,4,2,1
          B - 2,1

    MACRO: SYSTEM

    *PIO* DOES NOT CHECK IF THE I/O IS COMPLETE FOR THE INDICATED
    FET. THIS IS ENTIRELY USER-S RESPONSIBILITY.

--  PAR  -  SET REGISTERS FROM PARAMS

    A1 IS EXPECTED TO POINT TO A SYMPL ADDRESS VECTOR FOR:

      FET [0],LFN,BUF,PRU NO.,NO. OF PRU-S,RECALL

    THE REGISTER ASSIGNMENT WILL SUIT *PIO*
```

```
--  NEXT CARD IS *CALL CDRECALLC

--  RECALLC  -

    RELINQUISH THE CPU UNTIL PP FUNCTION IS COMPLETED

    ENTRY: X1 = FET ADDRESS

    MACRO: RECALL

--  RSRASSC  -

    RESET/SET *RA.SSC* UNLESS BIT *S$DROP* HAS BEEN SET BEFORE
    FOR SUBSYSTEM TERMINATION OR BIT *S$FILREQ* IS SET IN WHICH
    CASE WE ARE TO PROCESS A REQUEST FROM THE REQUEST FILE AND
    THE FREE UCP BUFFER IS USED ANYWAY.

    USES: A - 6
          X - 6,2

    MACRO: IFBIT

    CALLS: UCPBADD

--  RTIME  -

    READ REAL TIME CLOCK

    USES: A - 6,1
          X - 6,1,0

    MACRO: RTIME

    EXIT: X6 = REAL TIME IN SECS, X0 = MASK (-24D)

--  SFENDTC  -

    INFORM UCP OF SUBSYSTEM TASK COMPLETION

    SYMPL CALLABLE:  SFENDTC (FLAG).

      FLAG  =  0    =>    ISSUE FUNCTION WITH *UCPA* > 0
                          (*UCPA* TAKEN FROM *AP+1*)
      FLAG  <>  0   =>    ISSUE FUNCTION WITH UCPA = -1

    FUNCTION 'SF.ENDT' WILL BE ISSUED UP TO *CT$SFRT* TIMES.
    IF NOT SUCCESSFUL *OV.SFEND* IS LOADED TO HANDLE THE
    REQUEST.

    USES: A - 7,6,5,2,1
          X - 7,6,5,4,2,1,0
          B - 3,1

    MACROS: RECALL, SFCALL, IFBIT

    CALLS: MEM, LOADGO, RECALLC

--  ISSUE 'SF.ENDT' FUNCTION TO UCP. *JN/JDT* AS WELL AS *UCPA*
    CAN BE EXTRACTED FROM THE UCP BUFFER WITH ADDRESS IN *UCP$BA*.

--  'SF.ENDT' NOT SUCCESSFUL FOR *CT$SFRT* RETRIES OR *SSF*
```

```
    ERROR RETURN CODE DIFFERENT FROM *SFRC$USO* WHICH MEANS
    UCP NOT SWAPPED OUT. WE WILL LOAD OVERLAY *OV$SFEND* TO
    PROCESS THE REQUEST. (NOTE: *OV$SFENDT* WILL OVERLAY THE
    CURRENT FUNCTION OVERLAY).

--  SYSEND  -

    GENERAL CLEAN-UP AT END-OF-PROGRAM

--  TERMESS  -

    SEND MESSAGE TO MESSAGE TERMINAL.

    SEND THE MESSAGE IN *M.BUF* TO THE INTERACTIVE TERMINAL USING
    *MES*. THE PARAMETER BLOCK *MES$PAR* IS EXPECTED TO BE SET
    PROPERLY WITH THE CORRECT TID FOR THE MESSAGE TERMINAL.
    IF *TERMESS* FAILS FOR THE FIRST TIME THE CENTRAL OPERATOR
    WILL BE INFORMED BY A NEW B-DISPLAY MESSAGE (CF. *BDIS2ND*).
    WE WILL JUST CLEAR BIT *S$MESS* AND CALL *BDIS2ND*.

    NOTE THAT SENDING MESSAGES TO INTERACTIVE TERMINALS USING
    *MES* IS JUST POSSIBLE AFTER A MODIFICATION MADE LOCALLY
    AT ECMWF. BECAUSE ONLY A LIMITED NUMBER OF OUTPUT BUFFERS
    CAN BE ALLOCATED BY *MES* (*MES* USES MACRO *OUTPP*) "PGM.SMS"
    CAN GET STUCK IF THE MESSAGE LOGGING TERMINAL IS ASYNCHRONOUS
    AND IN SENDING MODE NOT FOLLOWED BY A CARRIAGE RETURN. *MES*
    WILL THEN GO INTO THE DELAY STACK AND JUST WAIT (NOTE THAT
    THE CALL IS AUTO-RECALL). TO AT LEAST GIVE THE OPERATOR A
    CHANCE TO GET HIS TERMINAL WORKING AGAIN A B-DISPLAY MESSAGE
    *M.TERM* IS ISSUED BEFORE TO CALL *MES*. THIS MESSAGE WILL
    STAY IN THE ABOVE SITUATION AND IS CLEARED OTHERWISE AFTER
    THE CALL.

    ENTRY: B1 = 1

    USES: A - 6,1
          X - 6,1,0
          B - 5

    MACROS: IFBIT, CLEARBIT, MESSAGE, RECALL, SYSTEM, WRITEC

    CALLS: BDIS2ND, GENMESS

--  SEND MESSAGE TO INTERACTIVE MESSAGE TERMINAL

--  *MES* FAILED -> CLEAR BIT *S$MESS* AND REFRESH B-DISPLAY

--  AND WRITE SOME NOTIFICATION INTO THE LOGFILE AS WELL

--  TERMIN  -

    EXIT FROM SCP STATUS / TERMINATE PROGRAM

    IF BIT *S$PEX* IS SET (I.E. *TERMIN* CALLED AFTER *PEX*) WE
    ALSO CHECK BIT *S$NOREQ* TO SEE IF WE HAVE TO PROCESS A LAST
    UCP REQUEST. IF SO WE WILL CLEAR *S$PEX* AND *S$MORFR*,
    PROCESS THE LAST REQUEST AND WILL COME TO *TERMIN* AGAIN
    (!! *S$DROP STILL SET) AND END NORMALLY.

    IF BIT *S$SCP* IS SET WE WILL CALL *SSF* TO CANCEL SCP
    STATUS. NORMAL TERMINATION WILL THEN INCLUDE AN ERROR
```

```
     MESSAGE WRITTEN TO THE LOGFILE IF BITS *S$EROD* OR
     *S$ERRN* ARE SET FOLLOWED BY A CALL TO *SYSEND* AND
     *ENDRUN*.

     USES: A - 7,6,1
           X - 7,6,1
           B - 7

     MACROS: CLEARBIT, ENDRUN, IFBIT, LOGMESS, SFCALL

     CALLS: SYSEND

--   EXIT FROM SCP STATUS. THE FOLLOWING ACTIONS ARE PERFORMED.

       1.  MAKE AN 'SF.EXIT' CALL
       2.  CLEAR *RA.SSID*
       3.  CALL *SYSEND*
       4.  ISSUE *ENDRUN* OR KILL JOB

--   UCPBADD -

     ALSO EXTERNALLY CALLABLE FUNCTION TO RETURN THE ADDRESS
     OF THE FREE UCP BUFFER.

     USES: A - 2
           X - 6,2
           B - NONE
```

6.7 Data Structures

All more complex (SYMPL) data structures have been defined using 'system independent arrays' (cf. SYMPL Ref. Man.). At the present moment there is no program available which will plot tables describing these data structures, but a procedure named VARLIST is contained within the SMS UPDATE OLDPL which will produce a line printer output of most data structures containing the variable name, the data type, the starting word and bit positions plus the length of each variable defined.

For some of the most common structures a description will be given here.

6.7.1 /FI/ - Family Index Definition

For each family defined in a suite a so-called Family Index entry is generated. Within the definition of /FI/ is also contained the header for the files STASx and STABx. The following allocation holds:

FI$	INT	0	0	60
FI$CHECK	INT	1	0	60
FI$LFN	CHAR	2	0	6
FI$CRDAT	CHAR	2	36	4
FI$OPDAT	CHAR	3	0	10
FIOPYY	CHAR	3	6	2
FIOPMM	CHAR	3	24	2
FIOPDD	CHAR	3	42	2
FI$STSUITE	UNS	4	0	11
FI$FILSIZE	UNS	4	11	10
FI$DECK	BOOL	4	21	1
FI$LAST	UNS	4	22	11
FI$FBENTLG	UNS	4	33	3
FI$PRU	CHAR	5	0	20
FI$FAMNAM	CHAR	7	0	5
FI$NEXTFA	UNS	7	30	10
FI$FBLAST	UNS	7	40	8
FI$FAMSTAT	UNS	7	48	12
FI$STBIT	BOOL	7	50	1
FI$PTBIT	BOOL	7	51	1
FI$ETBIT	BOOL	7	52	1
FI$TBIT	BOOL	7	53	1
FI$RBIT	BOOL	7	54	1
FI$FBIT	BOOL	7	55	1
FI$SBIT	BOOL	7	56	1
FI$EBIT	BOOL	7	57	1
FI$ABIT	BOOL	7	58	1
FI$CBIT	BOOL	7	59	1
FI$STARTIM	UNS	10	0	11
FI$ENDTIM	UNS	10	11	11
FI$RESTIM	UNS	10	22	11
FI$UNUSED2	UNS	10	33	3
FI$PRUTT	UNS	10	36	12
FI$PRUBUP	UNS	10	48	12

/FI/ is sorted alphabetically according to the family name (FI$fAMNAM) in order to be able to perform binary searches. The original order is preserved through forward links (FI$NEXTFA).

6.7.2 /FE/ - Family Event Table

The Family Event Table contains all families which have a ´TRIGGER´ parameter associated with them. Each of these family entries is followed by an entry for each (family name, event) pair in the ´TRIGGER´ list.

/FE/ is stored following /FI/ on STASx and STABx.

FE$FAMNAM	CHAR	0	0	5
FE$TRIGGER	BOOL	0	30	1
FE$BITMAP	UNS	0	31	14
FE$EVENTNA	UNS	0	31	17
FE$LAST	UNS	0	48	12

6.7.3 /FB/ - Family Buffer

A Family Buffer of variable length is generated for each family the random addresses of which on STASx and STABx respectively are stored in /FI/. Each ´TASK´ entry and each ´FAMILY´ entry within a family definition occupy one Family Buffer entry:

FB$TASKFAM	CHAR	0	0	5
FB$JN	CHAR	0	0	7
FB$BITMAP	UNS	0	42	9
FB$NOREST	BOOL	0	51	1
FB$STATUS	UNS	0	51	9
FB$TIBIT	BOOL	0	52	1
FB$OBIT	BOOL	0	53	1
FB$DBIT	BOOL	0	54	1
FB$FBIT	BOOL	0	55	1
FB$SBIT	BOOL	0	56	1
FB$TBIT	BOOL	0	57	1
FB$ABIT	BOOL	0	58	1
FB$CBIT	BOOL	0	59	1
FB$STARTIM	UNS	1	0	11
FB$MCTYPE	UNS	1	11	3
FB$DATEBIT	BOOL	1	14	1
FB$WEEKBUF	UNS	1	15	7
FB$DATEBUF	UNS	1	15	45
FB$DECKPRU	UNS	2	0	12
FB$EVSPACE	CHAR	2	12	18

The Family Buffers /FB/ are copied behind /FE/ on STASx. Only families which have been started and are not yet complete will have a Family Buffer on the work file STABx. The position is random and is determined during execution. The length of a Family Buffer is recorded in the corresponding family entry in /FI/ (FI$FBLAST).

6.7.4 /DT/ - Delay Table

The so-called Delay Table occupies the space behind /FE/ on the work file STABx. Delay table entries are generated for families or tasks with start times ahead of the times when their S-Bits are being set.

DT$NEXTDAY	BOOL	0	0	1
DT$INT	INT	0	0	60
DT$FAMFLAG	BOOL	0	1	1
DT$TIME	UNS	0	2	11
DT$FI	UNS	0	13	10
DT$FB	UNS	0	23	8

6.8 Installation Notes

The entire SMS Subsystem source code is contained in an UPDATE OLDPL program library. This library is structured in sections containing source codes for the different compilers being used. The beginning of a new section is identified by a new deck name of the form ´*xyz´ where ´xyz´ is a mnemonic for the section to follow. The current section identifiers are:

*CC-DECKS Standard NOS/BE Common Common Decks plus some non-standard ones in use by the subsystem.

*COMDECKS All UPDATE COMDECKS being used.

*CALTEXTS CAL texts.

*CFT CFT subroutines.

*CFTPROGS CFT programs.

*COMTEXTS COMPASS texts.

*SYMTEXTS SYMPL texts.

*COMPASS COMPASS subroutines and preset type definitions.

*SYMPL SYMPL subroutines.

*CAPSULES CAPSULES being used by the UCP interface.

*C-PROGS COMPASS main programs.

*S-PROGS SYMPL main programs.

*INSTJOB Installation jobs for CYBER and CRAY production and test versions.

*PROCS CCL procedures to be used in conjunction with the SMS subsystem.

*TESTPRGS For test programs.

The convention has been adopted that within each of the above section all decks are alphabetically ordered. Furthermore, COMPASS and CAL deck names always have a ´C´ appended to them and all CRAY decks have a ´-C´ string appended as well. Decknames always symbolise the program unit names. (Ex.: Deck TTGENC contains the COMPASS main program of the task table generator while deck TTGEN contains the SYMPL part of the program). COMPASS preset decks have always got a ´=´ appended, eg. deck SMSFDL= contains presets for routine within deck SMSFDL.

The installation jobs within the SMS UPDATE program library rely on the the structure described above (note, that all of the ´*...´ decks contain ´*WEOR´ cards; so, decks are expected to be at the right place).

An interactive CCL procedure ´SMSI´ is contained within the SMS UPDATE library which when excecuted performs an update of the existing installation jobs for the CYBER and CRAY versions of SMS in the same program library to install a new version of either the CRAY or CYBER part. In case of the CYBER installation the subsystem is built from the so-called user libraries (cf. NOS/BE Installation Handbook: User Library Method) and EDITLIBed into appropriate libraries for subsequent deadstart tape generations; for CRAY versions an installation run is performed using the standard products and libraries: a new CRAY BUILD library is generated which is subsequently disposed via the CYBER/CRAY link in transparent format to the NOS/BE install private device set from where it can subsequently be copied etc.

I N D E X