
British Standard Specification for
Industrial real-time FORTRAN

[ISO title: Industrial real-time FORTRAN — Application for the control of industrial processes]

Contents

	Page		Page
National foreword	Inside front cover	Section three. Process input/output	
Committees responsible	Back cover	10 Introduction	18
0 Introduction	1	11 Scope of the process I/O and general structure of I/O routines	18
1 Scope and field of application	1	12 Input/output of analog values	18
2 Definitions	1	13 Input/output of digital values	19
Section one. Multiprogramming and real-time features		Section four. File handling	
3 Introduction	3	14 Introduction	20
4 Date and time information	3	15 Background information	20
5 General aspects of tasking	3	16 File system environment	20
6 Procedure references	6	17 Procedures to control file access	21
Section two. Binary pattern and bit processing		Annexes	
7 Introduction	16	A Historical background	23
8 Binary pattern processing	16	B Further descriptions pertaining to individual clauses	26
9 Bit processing	17	C Abortion of tasks	30
		D File handling	31
		Bibliography	32

National foreword

This British Standard has been prepared under the direction of the Information Systems Technology Standards Committee. It is identical with ISO 7846-1985 'Industrial real-time FORTRAN — Application for the control of industrial processes' published by the International Organization for Standardization (ISO). It is complementary to ANSI X3.9-1978 (technically equivalent to ISO 1539-1980), the internationally recognized standard for the computer language FORTRAN, and needs to be read in conjunction with that standard.

BSI withdrew from participation in the work on the preparation of ISO 7846-1985 when it became apparent that, at that time, the level of UK interest in the project did not provide an adequate basis for participation. However, the responsible Technical Committee IST/5, has reviewed the ISO text, and has approved it as suitable for publication as a British Standard without deviation.

Terminology and conventions. Some terminology and certain conventions are not identical with those used in British Standards; attention is drawn especially to the following.

Wherever the words 'International Standard' appear, referring to this standard, they should be read as 'British Standard'.

Cross- reference

International standard	Corresponding British Standard
ISO 2382/10-1979	BS 3527 Glossary of terms used in data processing Part 10 : 1979 Operating techniques and facilities (Identical)

ISO 1539-1980 'Programming languages — FORTRAN', to which reference is made in the text, is the endorsement by ISO of the ANSI national standard X3.9-1978.

A British Standard method for specifying requirements for FORTRAN language processors is currently being prepared for publication. It is complementary to the ANSI standard X3.9-1978 and therefore to ISO 1539-1980.

Copies of ANSI X3.9-1978 can be obtained from the Sales Department, BSI, Linford Wood, Milton Keynes MK14 6LE.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

British Standard Specification for

Industrial real-time FORTRAN

0 Introduction

This International Standard describes a task model and a set of related routines to allow control of multi-task systems using FORTRAN as the programming language. Information concerning the development of this International Standard is given in annex A, and suggestions and justifications for some features are given in annex B. Annex C deals with the problem of aborting tasks and annex D with file handling. These annexes do not form part of this International Standard.

1 Scope and field of application

This International Standard establishes external procedure references for use in industrial computer control systems. These external procedure references provide access to time and date information, permit interface of programs with executive systems, process input and output functions, allow manipulation of bit strings and provide a method for file handling.

These procedures are intended for use with programs written in FORTRAN in accordance with ISO 1539.¹⁾ These programs are expected to be executable both in a solitary and in a multiprogramming environment under the control of a real-time executive system.

It is the responsibility of the executive system to deal with exceptions or errors, such as division by zero or referencing $\text{SQRT}(-1.0)$, so that such errors do not cause serious effects, for example unwanted or uncontrolled abortion, or affect other tasks.

This International Standard is applicable to all FORTRAN systems which require multi-tasking features.

NOTE — No extensions or variations from this International Standard should be implemented except for features explicitly declared in this International Standard to be processor dependent.

2 Definitions

For the purpose of this International Standard, the following definitions apply.

NOTE — Terms printed in italics in the definitions are themselves defined in this clause. However, in each definition, such items are only printed in italic at their first occurrence.

2.1 access mode: The right or permission to access (read or write) a *file* granted by the *processor* following a request for such permission.

2.2 basic clock counts: Counts of the basic unit of the system real-time clock as available for the user's program.

2.3 computation: A set of *operations* carried out on a set of data, as available for the user's program.

2.4 critical region: A part of a *sequential order of operations* operating on shared data such that this part of the program must have exclusive access to the shared data during the *execution*.

2.5 DORMANT: One of the definite states of a *task*.

A dormant task is known to the *executive system* and is not in any of the states *PENDING*, *RUNNING*, or *SUSPENDED*.

2.6 event: A significant discrete occurrence or incident which is intended to affect the *execution* of some *task* in a planned manner.

An event itself occurs instantaneously and sets an *eventmark*¹⁾.

2.7 eventmark: Internal variable of the *executive system*, used to indicate that an *event* has occurred.

If the user's system contains *parallel tasks*, the eventmarks are shared data elements for the *tasks*.¹⁾

2.8 execution: The collection of actions performed by a computer *processor* carrying out instructions in a sequential manner.

2.9 executable program: A program including all of its functions and subroutines in a form suitable for *execution*.

1) See also 5.3.1.

2.10 executive routine; executive system: The part of the *processor* which supports the procedures described in this International Standard.

2.11 file: A collection of related records treated as a unit.

For the purpose of this International Standard, the records are considered to be of fixed length. Record storage and access are independent of the internal format of records.

2.12 initiation: The action taken by the *executive system* to start the *execution* of a *task* at its first executable statement.

2.13 multiprocessing¹⁾: A mode of *operation* that provides for parallel processing by two or more *processors* of a multiprocessor.

2.14 multiprogramming¹⁾: A mode of *operation* that provides for the interleaved *execution* of two or more computer programs by a single processor.

2.15 multitasking¹⁾: A multiple *operation* that provides for the concurrent performance or interleaved *execution* of two or more *tasks*.

2.16 NON-EXISTENT: One of the definite (formal) states of a *task*.

A non-existent task is unknown to the *executive system*.

2.17 object task; designated task; referenced task: The *task* that is wanted or expected to be started, halted, stopped, or otherwise affected as a consequence of a system subroutine call.

2.18 operation: A deterministic rule for the generation of a finite set of data from another finite set of data.

2.19 overrun: Occurs when the condition for *initiation* of a *task* becomes true while the task is still running because of a previous *initiation*.

2.20 parallel tasks; concurrent tasks: A set of *tasks* whose *operations* may overlap in *time*.

2.21 PENDING: One of the definite states of a *task*.

A pending task has been associated with an *event* or *time* condition such that when the condition occurs, the task will be transferred to state *RUNNING* and initiated.

2.22 processor: The combination of a data processing system and the mechanism by which programs are transformed for use on that data processing system.

2.23 processor dependent: Describes an action of the *processor* that is not specified in this International Standard.

2.24 repetitive execution: Occurs when a *task* is repeatedly initiated, whether at fixed intervals or by repetitive *events*.

2.25 resourcemark: Internal variable of the *executive system*, used to indicate that a resource is exclusively reserved for a *task*.²⁾

2.26 RUNNING: One of the definite states of a *task*.

A running task is executing in its *virtual processor*.

2.27 semaphore: A variable of the *executive system*, used for the exchange of synchronizing information between interacting parallel *tasks*.

All semaphore *operations* in this International Standard imply *critical region* protection, provided by the executive system.

2.28 sequential order of operations: An order of *operations* the results of which are as if the operations are performed strictly one after another.

2.29 SUSPENDED: One of the definite states of a *task*.

A suspended task has temporarily halted the *execution* of its *virtual processor*, and is waiting for a specified condition to continue the execution of its virtual processor.

2.30 task: A *computation* which can be scheduled.

The *operations* of this computation are performed in a strict *sequential order of operations*.

(See also 2.17 and 2.28.)

2.31 time:

- a) **absolute time:** Complete time and date specification.
- b) **relative time:** A time increment or difference.

2.32 virtual processor: An environment in which a *task* can run from the time it is initiated until it terminates without consideration of availability of resources, managed by the *processor* and not by the user's program.

A particular implementation serves to map a set of virtual processors onto a set of real processor(s). This mapping is *processor dependent*.

¹⁾ Definition taken from ISO 2382/10.^[2]

²⁾ See also 5.3.2.

Section one: Multiprogramming and real-time features

3 Introduction

This section describes several procedure references available for the user's program, and relating to multiprogramming and, in particular, real-time operation. For all calls given in this section, the operations are generally considered indivisible, i.e. their operation will behave as if they are not interrupted.

4 Date and time information

For programming in a real-time environment, the user must have access to the time variables of the executive system. These time variables are obtained by system calls as described below.

Unambiguous time specification requires unique designation of time including complete date and an acknowledged calendar, defining time zero. Execution of reference to subroutine DATIM provides this complete information. The date refers to the Gregorian calendar.

The calls are

CALL DATIM(t1) for obtaining current date and time,
CALL CLOCK(j,k1,k2) for obtaining the basic clock counts.

4.1 Obtain date and time

The form of the call is

CALL DATIM(t1)

where t1 designates an integer array, into whose first 8 elements will be placed the absolute time, as expressed by the executive system's real-time clock at the time when the call is executed. These elements are as follows :

first element: Counts of the basic clock
second element: Milliseconds (0 to 999)
third element: Seconds (0 to 59)
fourth element: Minutes (0 to 59)
fifth element: Hours (0 to 23)
sixth element: Day (1 to 31)
seventh element: Month (1 to 12)
eighth element: Year

4.2 Obtain clock counts¹⁾

Execution of a reference to this subroutine allows the user program to obtain the current value of the system real-time clock, expressed in basic clock counts.

The form of the call is

CALL CLOCK(j,k1,k2)

where

j designates an integer variable or integer array element into which the current value of the clock will be placed as a positive integer. j is counted up to a maximum value as given by k2, then set to zero and counted again;

k1 is the number of basic clock counts per second. It is an integer value returned by the system. This argument shall be an integer variable or an integer array element;

k2 is the maximum value j can attain. It is an integer value returned by the system. This argument shall be an integer variable or an integer array element.

The modulus size of j is given by argument k2 (modulus = k2 + 1). In practice, this will usually be equal to the modulus of the hardware counter realizing the clock.

5 General aspects of tasking

5.1 States and transitions

At any time, a task is in one and only one state. Actions executed by the executive system, other tasks, or the designated task itself, may cause transition from one state to another. These transitions are performed instantly, i.e. they are considered ideally to take no time.

This mathematical model of a task may be visualized by a "state diagram", such as that shown in the figure, in which the states are nodes, illustrated as circles, while transitions are drawn as pointed arrows from one node to another.^[3]

¹⁾ See also annex B.

A multitasking system consists of several parallel tasks and can be considered as modelled by a number of disjointed but similar diagrams. It is feasible to apply a three-dimensional picture, with the similar diagrams sandwiched on top of each other and oriented so that the identical states of the individual diagrams cover each other.

State transitions are generally caused by subroutine calls, occurrence of events, or expiration of time limits. The name, form, and interpretation of the subroutine calls are standardized and described below.

In this International Standard, a task is described by a mathematical model illustrated by the state diagram shown in the figure. This model adheres to the following basic principles:

- transitions are non-ambiguous, i.e. for a given stimulus in a given state, the task can transit to only one possible new state;
- transitions are performed instantly, i.e. in zero time;
- a task exists in one state only at a time;
- the state model describes the behaviour of tasks as seen by the application programmer.

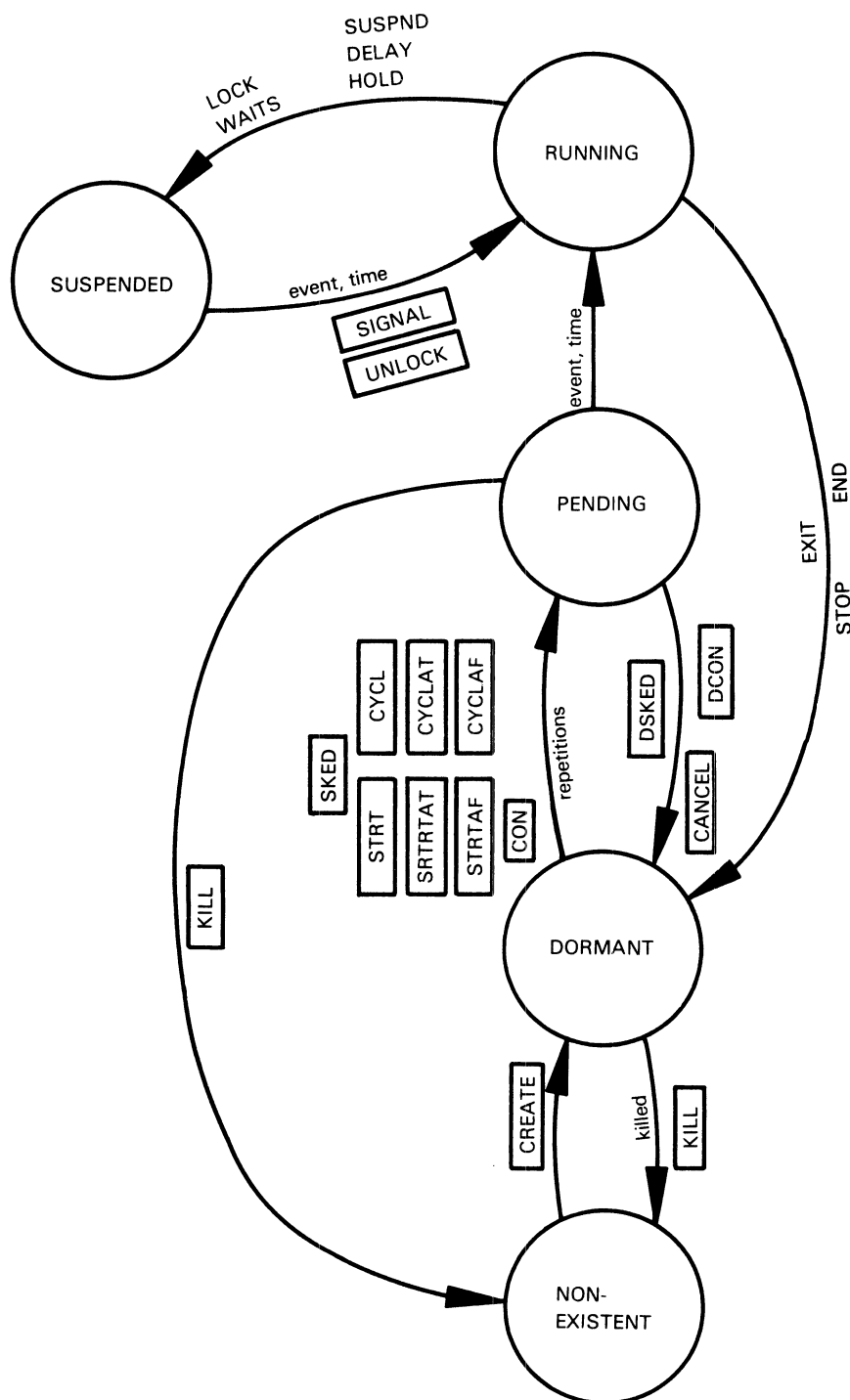


Figure — State model and transition diagram

The following symbolism is used for transitions in the state diagram:

- Capital letters in box : Effect on a task imposed by another task, i.e. a subroutine call in one task has the indicated effect on the designated task.
- Capital letters without box : Effect imposed by the task itself while in the state RUNNING.
- Small letters : Conditions under which the executive system performs the indicated state transitions.

With reference to principle d) above, no attempt is made to describe executive system actions transparent to the application programmer. Consequently, the state RUNNING is related to the task's virtual processor (see 2.32). It is immaterial for the state of a task whether a physical processor happens to be assigned to it or the execution is temporarily hampered by the executive system due to limited availability of physical processors and the task's low priority. Thus, the model is as well adapted to multiprocessors as it is to single processor computers.

5.2 Multiple activation calls¹⁾

Different tasks may issue apparently conflicting transition calls for the same object task. This will be the case during normal operation, as well as under error conditions, since the state of an object task is unknown at the time a transition call is made by another task.

In accordance with the principle of a task being in one state only at a time (see 5.1), a distinction is made between state transitions and calls for such transitions. Transition calls are received by the executive system, which will apply its own scheduling strategy in handling such calls.

Depending on available resources, such as internal table space, etc., of the executive system, a transition call will be accepted or rejected. An argument will be returned after the reference, with an appropriate value indicating whether the reference has been accepted normally or rejected.

5.3 Synchronization concepts

This International Standard provides three concepts for synchronization between tasks and the resolution of resource contention, as follows:

- eventmarks;
- resourcemarks;
- semaphores.

Eventmarks and semaphores are mainly used for synchronization purposes, whereas the resourcemarks are mainly used for the resolution of resource contention.

Eventmarks, resourcemarks, and semaphores are local variables of the executive system and they may not be accessed other than by the mechanisms described in this International Standard.

5.3.1 Eventmarks

In the management of concurrent tasks, it is necessary to associate certain tasks with certain events. These events may be either external or internal events. External events are of some physical nature, such as a contact closure, but the connection between an external event and its eventmark is beyond the scope of this International Standard. An internal event arises from specific program action (see 6.9.1 for a description of CALL POST).

Eventmarks are selected by reference to a numeric selector in the range 1 (one) to n, where n is processor dependent. Eventmarks have two states

ON
OFF

The association of events to tasks is done by the subroutine calls CALL SKED (see 6.4), CALL CON (see 6.5.8), CALL SUSPND (see 6.7), and CALL HOLD (see 6.8.1).

An eventmark is turned to ON by the occurrence of an internal or external event. If one or more tasks are associated with this event, the executive system will cause each associated task to begin or continue execution. An eventmark is turned to OFF by direct program control or by the executive system as it services the tasks associated with the eventmark. Eventmarks can be changed only by the procedure references defined in this International Standard and by the executive system that services the events.

Eventmarks may also be set to the OFF state by a specific program action, the CALL CLEAR (see 6.9.2), and they may be masked and unmasked (see 6.9.4 and 6.9.5). Additionally, the value of an eventmark may be tested by the logical function TESTEM (see 6.9.3).

5.3.2 Resourcemarks

A simple means to resolve contention for various resources is provided by the resourcemark concept, which permits one and only one task to use a resource at a time.

Resourcemarks are selected by reference to a numeric selector in the range 1 (one) to n, where n is processor dependent. Resourcemarks have two states

LOCKED
UNLOCKED

This International Standard does not define what can be considered to be a resource. It is the responsibility of the user to associate a resource with a resourcemark. If the user wants to reserve a resource exclusively for the running task, he chooses a resourcemark for the resource and performs the reference CALL LOCK (see 6.10.1). Should the resourcemark be in the state UNLOCKED at this moment, the resourcemark will change to state LOCKED, reserving the corresponding resource exclusively for this task. This reservation is later released by the task, by execution of a reference to EXIT (see 6.12) or UNLOCK (see 6.10.2).

1) See also annex B.

Should a task attempt to LOCK a resource mark which is already locked, the executive system will transfer this task into the state SUSPENDED, where it remains until the resource mark becomes unlocked by some other task.

If several tasks are waiting for a resource mark to be UNLOCKED, only one will be selected for execution by the executive system. (For details, see 6.10.2.) This is a main difference from the eventmark concept, where all tasks waiting for an eventmark are transferred to state RUNNING if the eventmark is set to ON.

A resource mark can also be set to LOCKED by reference to the logical function TLOCK (see 6.10.3).

5.3.3 Semaphores

Semaphores represent a synchronization concept useful for cases where a more advanced mechanism than those available by resource marks and eventmarks is desired. Semaphores are selected by reference to a numeric selector in the range 1 (one) to n , where n is processor dependent. Unlike eventmarks and resource marks, a semaphore has an integer value, s . The value s of a semaphore must be initialized and may later be set by the reference CALL PRESEM (see 6.11.1).

If the task needs to wait until the value of a semaphore is greater than or equal to a specific value, the task uses the reference CALL WAITS (see 6.11.2). By execution of CALL WAITS, the executive system tests if the specified decrement j is greater than s . In this case, the calling task is transferred to state SUSPENDED, where it remains until the semaphore value s is incremented by another task to a value greater than or equal to j . If j is not greater than s , the value s is decremented by j and the calling task continues its execution.

By execution of CALL SIGNAL (see 6.11.3), the executive system increments by j the value s of the specified semaphore. This incrementation may bring a task, waiting for this semaphore, from state SUSPENDED to state RUNNING, if s becomes greater than or equal to j of the suspended task. If multiple tasks are waiting for this semaphore and are candidates for running, after the incrementation of s by j , the order in which these tasks transit to state RUNNING is processor dependent.

By setting s to certain values and choosing different values for j , a variety of advanced synchronization concepts and solutions for resource contention are possible.

In addition to the calls described above, the value of a semaphore may be read using the integer function IRDSEM (see 6.11.4).

6 Procedure references

6.1 Terms and summary of procedure references¹⁾

This clause contains a summary of the subroutine calls and function references described in subsequent clauses.

The following designations for parameters apply to several of the calls. If the exact meaning of these parameter designations deviates from that described below, it will be marked specifically in the detailed description of the call. If the meaning is exactly as defined here, the description of the parameter will be omitted in the description of the call and a reference will be made to this subclause.

i specifies the task to be affected (object task). The argument shall be an integer array.

The contents of i may be partly generated by CREATE. i is used as input parameter in all other calls.

t, t_1, t_2 designate integer arrays, whose first 8 elements contain a specification of absolute or relative time. Negative values of elements are not permitted. These elements are as follows :

first element: Basic clock counts

second element: Milliseconds

third element: Seconds

fourth element: Minutes

fifth element: Hours

sixth element: Day(s)

seventh element: Month(s)

eighth element: Year(s)

If, for absolute times, value 0 is used for one of the three date elements, this shall be interpreted as "current date", "current month", or "current year" by the executive system.

The interpretation of a relative time specification containing months or years different from zero is processor dependent.

m is set on return to the calling program, to indicate the disposition of the request as follows :

0 or less: undefined

1: request accepted

2 or greater: request rejected (error condition)

This argument shall be an integer variable or integer array element. The processor may define specific values greater than or equal to 2 to distinguish between certain reasons for rejection.

The list of function and subroutine calls, described in detail in subsequent subclauses, is given in table 1.

¹⁾ See also annex B.

Table 1

Full description in subclause	Call	Parameters
6.2	CALL CREATE(i,m)	i : identification of created task and associated program
6.3	CALL KILL(i,m) (opposite of CREATE)	
6.4	CALL SKED(i,s,e1,t1,t2,e2,m) (general scheduling)	s : mode selector e1 : eventmark reference t1 : absolute or relative time for first initiation t2 : time period for cyclic initiations e2 : reference to eventmark for overrun
6.5.1	CALL STRT(i,m) (start immediately)	
6.5.2	CALL STRTAF(i,t1,m) (start after time delay)	t1 : time delay before initiation
6.5.3	CALL STRTAT(i,t1,m) (start at absolute time)	t1 : absolute time for initiation
6.5.5	CALL CYCL(i,t2,m) (cyclic, with immediate first initiation)	t2 : length of time interval
6.5.6	CALL CYCLAF(i,t1,t2,m) (cyclic, with delayed first initiation)	t1 : time delay before first initiation t2 : length of time interval
6.5.7	CALL CYCLAT(i,t1,t2,m) (cyclic, with absolute time specification of first initiation)	t1 : absolute time for first initiation t2 : length of time interval
6.5.8	CALL CON(i,e,m) (establish event connection)	e : eventmark reference
6.6	CALL DSKED(i,s,e,m) (eliminate scheduling)	s : mode selector e : eventmark reference
6.6.1	CALL DCON(i,e,m) (eliminate event connection)	e : eventmark reference
6.6.2	CALL CANCEL(i,m) (eliminate time scheduling)	
6.7	CALL SUSPND(s,e,t,n,m) (suspend continuation of calling task for time period or until event)	s : mode selector e : reference to eventmark for end of delay t : time delay n : indicator for cause of end of delay
6.8.1	CALL HOLD(e,m) (suspend until event occurs)	e : reference to eventmark for end of delay
6.8.2	CALL DELAY(t,m) (suspend for a relative time)	t : time delay
6.9.1	CALL POST(e,m) (setting of an eventmark)	e : eventmark reference
6.9.2	CALL CLEAR(e,m) (resetting of an eventmark)	e : eventmark reference
6.9.3	TESTEM(e,m) (testing the state of an eventmark)	e : eventmark reference function value : condition of the eventmark
6.9.4	CALL MKEM(e,m) (setting the mask of an eventmark)	e : eventmark reference
6.9.5	CALL UNMKEM(e,m) (clearing the mask of an eventmark)	e : eventmark reference
6.10.1	CALL LOCK(r,m) (locking of a resource mark)	r : resource mark reference
6.10.2	CALL UNLOCK(r,m) (unlocking of a resource mark)	r : resource mark reference

Table 1 (concluded)

Full description in subclause	Call	Parameters
6.10.3	TLOCK(r,m) (testing and locking of a resource mark)	r : resource mark reference function value : condition of the resource mark
6.11.1	CALL PRESEM(r,s,m) (initialization of semaphore)	r : semaphore reference s : initial value of semaphore
6.11.2	CALL WAITS(r,j,m) (wait on semaphore)	r : semaphore reference j : decrement
6.11.3	CALL SIGNAL(r,j,m) (release semaphore)	r : semaphore reference j : increment
6.11.4	IRDSEM(r,m) (read semaphore value)	r : semaphore reference function value : value of semaphore variable
6.12	CALL EXIT (termination of execution)	

6.2 Creation of a new task

A new task is introduced to the real-time system by reference to subroutine CREATE. The designated task will be associated with some specified program, considered a resource like other resources, necessary for the task to perform. The associated program is normally assumed to exist in an executable form. Formally, and in terms of the state model, the task is transferred from NON-EXISTENT to DORMANT as the effect of the reference (see the figure and 5.1).

A mechanism is assumed to exist outside the scope of this International Standard to create and initiate at least a first task, i.e. the parent, which in its turn may create other tasks.

The form of the call is

CALL CREATE(i,m)

where

i specifies an integer array which contains all the information necessary to specify the task and its associated program. The latter includes, among other items, its designation, where the program can be found (such as description of file), residency while existent (primary memory resident or swappable), etc. The array may also contain the task's processor priority.

This array will in general also contain output information, i.e. references distinguishing this task from other tasks created from an identical program code, reference to individual data sets, etc.

Such information may be used by other procedure references in clause 6. For a description of parameter i, see 6.1.

All details of this array are processor dependent.

m is as described in 6.1.

6.3 Eliminating a task from the real-time system¹⁾

A reference to subroutine KILL will eliminate a designated task from the real-time system, by transferring it to state NON-EXISTENT. If the designated task is in state DORMANT or PENDING, the effect shall be carried out immediately. If the designated task is in state RUNNING or SUSPENDED, the termination shall affect only future executions. Thus, the designated task will continue its present execution without any intervention by this call.

The form of the call is

CALL KILL(i,m)

where i and m are as described in 6.1.

6.4 Scheduling a task

Execution of a reference to the subroutine SKED, or its derivatives as listed in 6.5, shall schedule the initiation of a designated task, establishing the condition for subsequent transition to RUNNING. If the designated task is in state DORMANT when the call is made, the designated task shall transit to state PENDING. If it is already in state PENDING, the reference shall augment its conditions for subsequent transfer to RUNNING according to the arguments of the call, such that the designated task will transit to RUNNING when any of the conditions still valid becomes true. The transition to state RUNNING requires the task being in state PENDING, otherwise an overrun condition occurs. The augmentation of running conditions is subject to any resource limitation of the processor, and any violation of such limitation will result in an error return.

When a task transits to RUNNING, the condition that caused this transition is removed from the possible complex of combined conditions, such that the other conditions remain. When a running task subsequently exits, by transition from RUNNING to DORMANT, possible scheduling conditions, remaining from previous scheduling calls, shall cause immediate transition to state PENDING. (Note, however, the definition of the term "overrun" and the provision for its indication, as explained below.)

1) See also annex B.

Each normally accepted reference to the subroutine SKED causes the following effect.

After expiration of a specified time delay or at a desired absolute time or upon the occurrence of a specified event, the object task is transferred to state RUNNING and begins at the first executable statement of the program. The actual time resolution obtainable in a specific industrial computer system is subject to the resolution of that system's real-time clock. If the object task is initiated by an event occurrence, the executive system will set the eventmark OFF, as part of the initiation. If more than one task is waiting for a specified eventmark to change to the condition ON, when the change occurs all these tasks will transit into state RUNNING.

The form of this procedure reference is

CALL SKED(i,s,e1,t1,t2,e2,m)

where

i and m are as described in 6.1.

s is an integer expression, specifying three categories of task scheduling (see below), as follows:

- the values of the argument s between 10 and 15 cause the task to be initiated once;
- the values of the argument s between 20 and 25 cause the task to be initiated periodically by time;
- the values of the argument s between 30 and 35 cause the task to be initiated whenever a specified eventmark becomes ON.

e1 specifies the eventmark for scheduling; it is an integer expression.

t1 designates an integer array containing the absolute, or relative, time for the scheduling. (The term relative time for scheduling means the time delay from the time the reference is executed until the intended running.) (See 6.1.)

t2 is an integer array to designate the time period for cyclic runs. (See 6.1.)

e2 is an eventmark for overrun. It will be turned ON if overrun occurs. The action performed with the object task is processor dependent.

The values of the argument s between 10 and 15 define the first and only execution of the task.

- If s = 10: start immediately
- If s = 11: start at absolute time t1
- If s = 12: start after time t1
- If s = 13: start at event e1 (once)
- If s = 14: start at absolute time t1 or event e1 (once)
- If s = 15: start after time t1 or at event e1 (once)

The values of the argument s between 20 and 25 define the first execution of the task which is subsequently executed repetitively at the time period t2.

- If s = 20: start immediately plus cyclic by t2
- If s = 21: start at t1 plus cyclic by t2
- If s = 22: start after t1 plus cyclic by t2
- If s = 23: start at e1 plus cyclic by t2
- If s = 24: start at t1 or e1 plus cyclic by t2
- If s = 25: start after t1 or at e1 plus cyclic by t2

The values of the argument s between 30 and 35 define the first execution of the task which is subsequently executed whenever the eventmark specified by e1 becomes ON.

- If s = 30: start immediately plus repeated by e1
- If s = 31: start at t1 plus repeated by e1
- If s = 32: start after t1 plus repeated by e1
- If s = 33: start at e1 plus repeated by e1
- If s = 34: start at t1 or e1 plus repeated by e1
- If s = 35: start after t1 or at e1 plus repeated by e1

6.5 Starting a task by simplified calls

The following calls for scheduling represent subsets of the features of CALL SKED. They are established for ease of programming only.

CALL STRT(i,m)	start immediately
CALL STRTAF(i,t1,m)	start after time t1
CALL STRTAT(i,t1,m)	start at time t1
CALL CYCL(i,t2,m)	start immediately plus cyclic by t2
CALL CYCLAF(i,t1,t2,m)	start after time t1 plus cyclic by t2
CALL CYCLAT(i,t1,t2,m)	start at time t1 plus cyclic by t2
CALL CON(i,e1,m)	start at event e1 plus repeated by e1

6.5.1 Starting a task immediately

Execution of a reference to subroutine STRT establishes a condition for immediate transfer of the designated task to RUNNING via PENDING. Execution begins at the task's first executable statement. If in state DORMANT when the reference is made, the task will transit to state PENDING and continue immediately to state RUNNING.

The form of the call is

CALL STRT(i,m)

where i and m are as described in 6.1.

6.5.2 Starting a task after a specified time delay

Execution of a reference to subroutine STRTAF establishes a time delay as the condition for transfer of the designated task to RUNNING via PENDING.

After expiration of a specified time delay after the time when the reference is executed, the designated task is expected to transfer to state RUNNING and will do so if its state at that time is PENDING. If in state DORMANT when the reference is