ECSS-E-70-32A
24 April 2006

Space engineering

Test and operations procedure language

ECSS Secretariat
ESA-ESTEC
Requirements & Standards Division
Noordwijk, The Netherlands

Published by: ESA Publications Division
ESTEC, P.O. Box 299,
2200 AG Noordwijk,
The Netherlands
ISSN: 1028-396X
Price: € 30
Printed in: The Netherlands.

Foreword

This Standard is one of the series of ECSS Standards intended to be applied together for the management, engineering and product assurance in space projects and applications. ECSS is a cooperative effort of the European Space Agency, national space agencies and European industry associations for the purpose of developing and maintaining common standards.

Requirements in this Standard are defined in terms of what shall be accomplished, rather than in terms of how to organize and perform the necessary work. This allows existing organizational structures and methods to be applied where they are effective, and for the structures and methods to evolve as necessary without rewriting the standards.

The formulation of this Standard takes into account the existing ISO 9000 family of documents.

This Standard has been prepared by the ECSS-E-70-32 Working Group, reviewed by the ECSS Engineering Panel and approved by the ECSS Steering Board.

(This page is intentionally left blank)

Contents

Figures

Tables

Introduction

The procedure is the principal mechanism employed by the end-user to control the space system during pre-launch functional testing and post-launch in-orbit operations.

This Standard identifies the requirements to be satisfied by any language used for the development of automated test and operation procedures.

It also defines a reference language that fulfils these requirements. This language is called the “procedure language for users in test and operations (PLUTO)”.

(This page is intentionally left blank)

1
Scope

This Standard specifies:

- The capabilities of the language used for the definition of procedures for space system testing and operations.
- The PLUTO language.

Clause 4 defines the context in which procedures operate.

Clause 5 contains the requirements for the procedure language.

Annex A specifies the PLUTO language. This includes:

- The "building blocks" that constitute procedures and the role that each of these building blocks plays in achieving the overall objectives of the procedure.
- The dynamic aspects of procedures i.e. the execution logic of each building block and execution relationships between these blocks.
- The syntax and semantics of the language itself.

Annex B specifies the engineering units to be supported by the procedure language.

Annex C specifies the mathematical, time and string functions to be supported by the procedure language.

(This page is intentionally left blank)

2
Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this ECSS Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this ECSS Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the publication referred to applies.

ECSS-P-001B ECSS - Glossary of terms

(This page is intentionally left blank)

3
Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ECSS-P-001 and the following apply:

3.1.1
activity

space system monitoring and control function

3.1.2
compound parameter

record comprised of any sequence of **reporting data**, arrays of **reporting data** and sub-records that are interpreted together

EXAMPLE An anomaly report generated by the space segment comprising an anomaly report ID and a set of associated **parameters**.

3.1.3
confirmation body

part of a **procedure** (or **step**) whose purpose is to assess whether or not the objective of the **procedure** (or **step**) has been achieved

3.1.4
continuation test

language construct used to define how the execution of a **procedure** (or **step**) proceeds after a constituent **step** (or **activity**) has been executed

3.1.5
event

occurrence of a condition or set of conditions that can arise during the course of a test session or a mission phase

3.1.6
initiation

act of requesting the execution of a **step** or an **activity**

3.1.7
main body

part of a **procedure** (or **step**) dedicated to achieving the objectives of the **procedure** (or **step**)

3.1.8
parameter

lowest level of elementary information that has a meaning for monitoring the space system

3.1.9
preconditions body

part of a **procedure** dedicated to ensuring that the **procedure** only executes if or when pre-defined initial conditions are satisfied

3.1.10
procedure

means for interacting with the space system in order to achieve a given objective or sequence of objectives

3.1.11
reporting data

data used for assessing the functioning of the space system

NOTE Reporting data can consist of a **parameter** (a simple type) or a **compound parameter** (a complex type).

3.1.12
space system model

representation of the space system in terms of its decomposition into **system elements**, the **activities** that can be performed on these **system elements**, the **reporting data** that reflects the state of these **system elements** and the **events** that can be raised and handled for the control of these **system elements**, **activities** or **reporting data**

3.1.13
statement

element of the **procedure** language which, together with other elements, implements the goal of a **procedure** (or **step**)

3.1.14
step

component of a **procedure** that achieves a well-defined sub-goal

3.1.15
system element

representation within the **space system model** of a functional element of the space system

3.1.16
watchdog body

part of a **procedure** (or **step**) which manages contingency situations that can arise during the execution of the **procedure** (or **step**)

3.1.17
watchdog step

component of the **watchdog body** dedicated to detecting the occurrence of a particular contingency condition and executing corrective actions

3.2 Abbreviated terms

The following abbreviated terms are defined and used within this Standard:

Abbreviation	Meaning
AIV	assembly, integration and verification
EBNF	extended Backus-Naur form
EGSE	electrical ground support equipment
EMCS	EGSE and mission control system
FCP	flight control procedure
FOP	flight operations plan
MMI	man-machine interface
PLUTO	procedure language for users in test and operations
SCOE	special check-out equipment
SSM	space system model

(This page is intentionally left blank)

4
Context of the procedure language

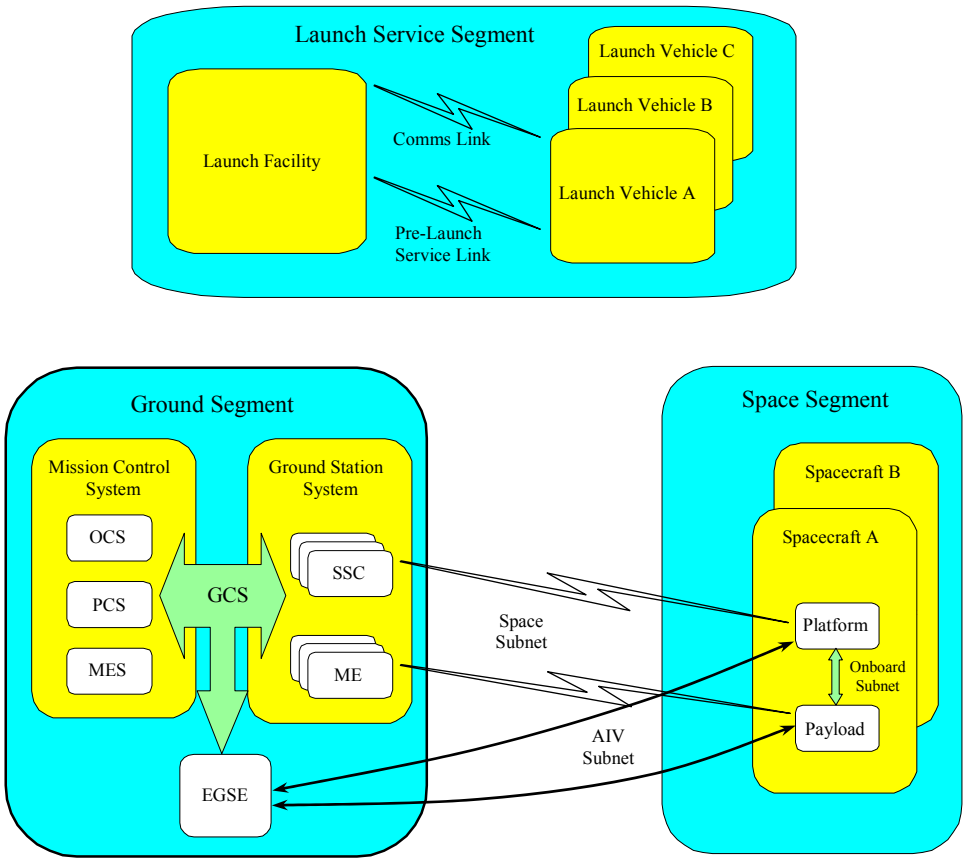
4.1 Introduction

4.1.1 The space system

ECSS-E-00 defines the overall space system as comprising a space segment, a ground segment and a launch service segment.

An example of the elements of a space system is shown in Figure 1. The space system elements shown in this figure are operational at different times:

- the electrical ground support equipment (EGSE) during the development phase;
- the launch service segment during the pre-launch and launch phases;
- the mission control and ground station systems during the mission operations phase.



Key: OCS: Operation control system
PCS: Payload control system
MES: Mission exploitation system
AIV: Assembly, integration and verification
SSC: Space segment control station
ME: Mission exploitation station
GCS: Ground communications subnet

Figure 1: Example of space system elements

4.1.2 Satellite testing

ECSS-E-10, ECSS-E-10-02 and ECSS-E-10-03 define the requirements for space system engineering, verification and testing.

This Standard does not prescribe the levels of integration and test at which procedures are used. This is considered to be a decision taken when the verification approach for a specific mission is defined. However, automated procedures are generally employed from the subsystem level upwards.

The re-use of test procedures at different levels of integration implies standardization of the functionality of the EGSE. Furthermore, the re-use of these procedures in the mission operations domain implies the harmonisation of the requirements for EGSE and mission control systems.

4.1.3 Mission operations

ECSS-E-70 identifies procedures as the primary mechanism for conducting mission operations and defines two types of flight control procedures (FCP):

- Nominal procedures

  These define the set of in-orbit operations of the space system to be used under nominal conditions. They constitute the building blocks from which the mission timelines and schedules of the flight operations plan (FOP) are constructed.

- Contingency procedures

  These define the recovery actions used to reconfigure the space system if pre-identified anomalies or failures occur.

Although FCPs have traditionally been executed under manual control, pressure to reduce manpower during routine mission operations implies more automation of routine tasks such as the execution of procedures.

4.2 EGSE and mission control system (EMCS)

4.2.1 General

In this Standard, the elements of the ground segment responsible for the monitoring and control of the overall space system, namely the EGSE and the mission control system are referred to jointly as the EMCS.

The procedure language and the procedure development and execution environments are an integral part of any EMCS. As such, they have direct access to other monitoring and control functions implemented within the overall EMCS.

4.2.2 Space system model

ECSS-E-70-31 introduces the concept of a space system model (SSM) as a means for capturing mission knowledge used during AIV and operations. This knowledge is used by the different EMCS applications in order to interact with the space system and to process the dynamic data that is exchanged with it (i.e. space segment telemetry and telecommands, ranging data, ground segment commands and measurements).

The SSM consists of different types of object and the relationships between these objects. The objects of relevance for the procedure language are system elements, reporting data, activities and events.

System elements correspond to:

- the elements of the space segment resulting from the functional decomposition defined in ECSS-E-00;
- the elements of the ground segment resulting from the functional decomposition defined in ECSS-E-70.

Reporting data and activities are associated with system elements:

- Reporting data comprises parameters and compound parameters. A parameter is the lowest level of elementary information that has a meaning for monitoring the space system. A compound parameter is a record comprised of any sequence of parameters, arrays of parameters and sub-records (see also ECSS-E-70-41). For example, a complete telemetry packet, or part thereof, may be represented as a compound parameter. The parameters within a compound parameter are normally interpreted together (e.g. when interpreting the contents of an anomaly report). Reporting data can have different representations depending on its life cycle within the space system (e.g. an on-board measurement has an encoded value in telemetry and a raw or engineering value when presented on a ground segment display).
- An activity is a space system monitoring and control function. The term activity is introduced to refer generically to procedures, telecommands (either to the space segment or to the ground segment) and any function provided by the underlying EMCS (e.g. a printer request, sending an e-

mail, transferring a file using ftp). A given mission can implement additional mission-specific activity types (e.g. conforming to a non-standard protocol).

Events are associated with system elements, reporting data and activities. An event is an occurrence of a condition or set of conditions that can arise during the course of a test session or a mission phase. It is used to trigger a monitoring and control function implemented within the space system.

An example of a space system model is shown in Figure 2.

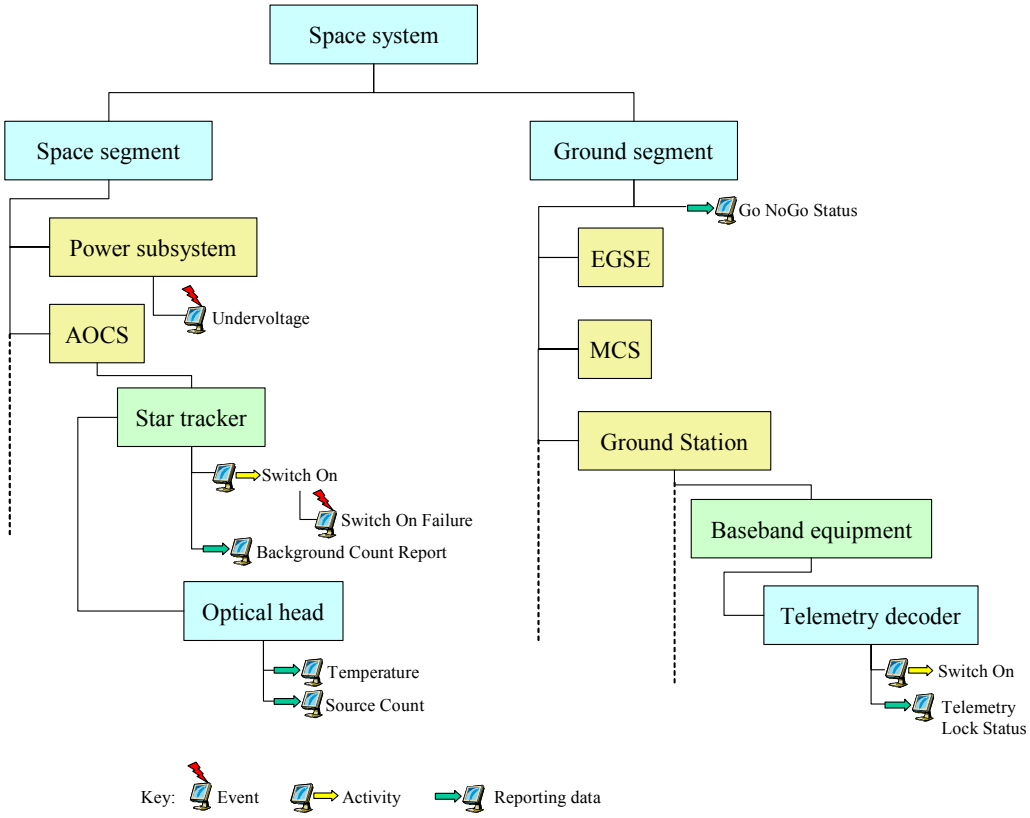


Figure 2: Example of a space system model

During the course of spacecraft testing and mission operations, the space system is configured in different ways, for example:

- to test elements of the space segment in a stand-alone manner;
- to test the space segment during integration when some elements are missing;
- to validate the ground segment with a simulated space segment.

The concepts of system elements, activities, reporting data and events provide a complete definition of the SSM independent of any specific space system configuration. However, for a given space system configuration, only a subset of the overall SSM is used.

In order to understand the scope of the procedure language, it is important to understand what is provided by the SSM. The SSM (as specified in ECSS-E-70-31) provides the definition of space system configurations, system elements, activities, reporting data and events.

The procedure language provides the means to:

- refer to system elements, activities, reporting data and events but not to define them;
- define the procedural script.

For activities, the common set of data definitions in the SSM includes:

- name;
- description;
- type (procedure, telecommand or operating system call);
- associated system element;
- version number and configuration history;
- validation status (i.e. information on the testing status of the activity);
- validity period, i.e. the earliest and the latest date on which the activity can be executed;
- expected (i.e. mean) duration;
- maximum and minimum durations;
- list of arguments, including for each argument:
  - name,
  - description,
  - engineering units,
  - data type, and
  - default value;
- allowed combinations of argument values;
- other attributes used for mission planning purposes such as the profile of resources that the activity utilizes (e.g. onboard power and downlink bandwidth).

In addition, the different types of activity have type-specific definitions; for procedures, this includes:

- default execution mode (manual or automatic);
- activation mode (i.e. whether the procedure is permanently active or is explicitly initiated);
- the procedural script.

(This page is intentionally left blank)

5
Requirements to be satisfied by procedures

5.1 Procedure structure

a. The capability shall be provided to construct a high-level, goal-oriented activity (namely a procedure) using elementary activities consisting of telecommands and operating system calls.

b. A procedure may include calls to execute other procedures.

c. The capability shall be provided to define self-contained sub-goals for a procedure.

d. A given sub-goal may be achieved by a single activity call or a sequence of activity calls.

e. An activity may have associated arguments whose values are passed to the activity at the time of initiation.

f. Where the achievement of a sub-goal involves complex logic (such as waiting for a period of time, waiting for a condition to become true or conditional branching), a step construct shall be provided to encapsulate the logic.

g. The capability shall be provided to execute procedure sub-goals in series or in parallel.

NOTE Parallel execution is used, for example, when two or more steps can be performed completely independently.

h. Preconditions may be specified for a procedure.

NOTE Preconditions are conditions to be fulfilled before the procedure can start.

i. Confirmation conditions may be specified for a procedure.

NOTE Confirmation conditions are those conditions that determine whether the goal of a procedure is met.

j. Preconditions and confirmation conditions may also be defined for steps.

k. In addition to the “nominal flow path” of a procedure (i.e. the flow designed to achieve its primary goal), contingency actions may also be defined.

l. A procedure contingency action may be executed upon:

1. detection of space system anomalies;
2. detection of a failure in the execution of the nominal path;
3. other specified events or conditions.

m. In addition to the “nominal flow path” of a step, contingency actions may also be defined.

n. A step contingency action may be executed upon:

1. detection of a failure in the execution of the nominal path;
2. other specified events or conditions.

o. A contingency action should be taken at the level at which a failure occurs, i.e. if a failure occurs in the execution of a step, a contingency action should be taken at the level of the step.

p. A contingency action should rectify the detected anomaly or report it to the user and return control to the nominal flow path of the procedure (or step).

q. If an anomaly cannot be rectified, one of the following actions shall be performed:

1. generate an event to flag the problem at a higher level for resolution;
2. use other means to achieve the goal of the procedure (or sub-goal of a step) and then terminate the procedure (or step);
3. abort the procedure (or step).

r. If a contingency action is invoked, the body of the procedure (or step) shall be suspended until the contingency action is completed.

5.2 Language constructs

a. The capability shall be provided to request the execution of any space system activity.

b. The capability shall be provided to request the execution of an activity and proceed immediately with the execution of the next statement of the procedure.

c. The capability shall be provided to request the execution of an activity and wait for the confirmation of execution before continuing.

d. When a procedure waits for the confirmation of execution of an activity, the result of the execution of the activity shall be tested in order to determine the subsequent course of action.

e. The capability shall be provided to acquire the following data:

1. the execution status or confirmation status of an initiated activity;
2. the initiation time, start time, termination time or completion time of the last execution of an activity;
3. the value of reporting data;
4. the validity status of a parameter;
5. the overall monitoring status or detailed monitoring status (limit-check, delta-check, expected status check or status consistency check) of a parameter;
6. the sampling time of reporting data;
7. the time of the last occurrence of an event.

f. When a given space system function is unavailable (e.g. during an early test phase), the capability shall be provided to replace the missing function. This includes setting:

1. the confirmation status of an activity;
2. the value of a parameter;
3. the validity status of a parameter;
4. the overall monitoring status or detailed monitoring status (limit-check, delta-check, expected status check or status consistency check) of a parameter;
5. the sampling time of reporting data.

g. To avoid the problem that can arise with asynchronous systems where a new instance of reporting data arrives between two successive procedure statements that use different properties of this object, the capability shall be provided to create a local copy of reporting data.

NOTE This ensures that the procedure uses properties of reporting data that have been sampled at the same instant in time.

h. To ensure that any service provided by the EMCS is accessible within a procedure, the following capabilities shall be provided:

1. to request any operation defined for any space system object;
2. to acquire the value of any property of any space system object.

i. The capability shall be provided to perform the following execution flow controls within a procedure:

1. simple conditional branching (i.e. if … then … else …);
2. multiple conditional branching where the path taken is dependent on the value of a specified parameter (or local variable, see j below);
3. repeated execution of a statement (or a group of statements) with the possibility of repeating the execution a specified number of times, repeating the execution indefinitely whilst a given condition holds true or repeating the execution until a given condition becomes true;
4. wait until a given absolute time;
5. wait for a given interval of time to elapse;
6. wait until a given condition becomes true;
7. wait until a given event occurs.

j. Local variables used within a procedure shall be declared and their type explicitly stated.

k. The capability shall be provided to assign a value to a local variable within a procedure.

l. The capability shall be provided to raise a local event within a procedure (e.g. to trigger a contingency action).

m. Local events shall be defined as part of the procedure (or step) declarations.

n. Mathematical, time and string functions shall be supported.

o. The capability shall be provided to construct expressions that operate on constants, space system parameters acquired from the EMCS, activity arguments (whose values are supplied at runtime) and local variables.

p. Engineering units shall be supported.

q. The capability to assign engineering units to constants shall be supported.

r. The capability to mix compatible units freely within an expression shall be supported

s. The automatic conversion between different, but compatible, units shall be supported.

t. Comments may be included within a procedure.

u. The capability shall be provided to generate a message for acknowledgement by the user.

v. The capability shall be provided to generate a message for entry in the procedure execution log.

w. The capability shall be provided to acquire inputs from the user.

x. The conditions specified for a procedure (or step) precondition or confirmation may be any combination of the following:

1. wait until a given absolute time;
2. wait for a given interval of time to elapse;
3. wait until a given condition becomes true;
4. wait until a given event occurs;
5. test whether a given condition is true;
6. request the user to specify the outcome.

5.3 Language specification

a. Any language that complies with this Standard shall be formally specified using the ISO extended Backus-Naur form (EBNF), see ISO/IEC 14977.

b. The PLUTO procedure language, engineering units and functions, as specified in annexes A, B and C should be used for the automation of procedures in the development and exploitation phases of space systems.

NOTE Use of the PLUTO language ensures conformance with all requirements of this Standard. It also facilitates the transfer of procedure knowledge, acquired in the development phase during functional testing, to the mission operations phase (for a given mission) and encourages the use of a common procedure language across missions.

Annex A (informative)
The PLUTO language

A.1 The structure of a procedure

A.1.1 Procedure definition

A procedure comprises the following elements:

a. An optional declaration body, which declares the local events that can be raised within the procedure.

b. An optional preconditions body, which ensures that the procedure is only executed if (or when) pre-defined initial conditions are satisfied.

c. A mandatory main body, which fulfils the goal of the procedure. The main body can be composed of self-contained sub-goals fulfilled by activities or steps.

d. An optional watchdog body, which manages contingency situations that can arise during the execution of the procedure. The watchdog body is composed of one or more special steps, called watchdog steps, which are all initiated in parallel.

e. An optional confirmation body, which assesses whether the objectives of the procedure have been achieved or not.

An example of a procedure and its elements is shown in Figure A-1.

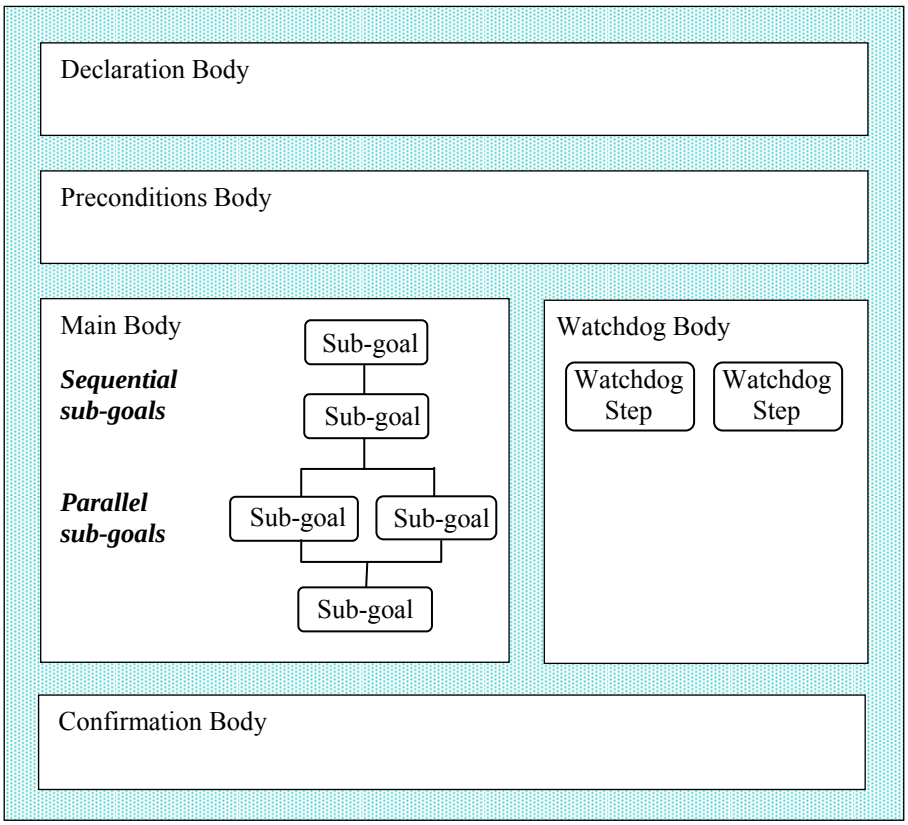


Figure A-1: Example of a procedure and its elements

A.1.2 Procedure declaration body

Local events that can be raised within the procedure, but which are accessible outside of the step in which they are raised (e.g. events that trigger a procedure-level watchdog, see A.1.5), are declared at procedure level.

Arguments may be passed to a procedure at initiation time by the entity calling the procedure. These arguments are defined externally (see subclause 4.2.2) and not as part of the procedure declarations.

Arguments are similar to parameters in their definition and use.

Argument values cannot be changed within a procedure.

A.1.3 Procedure preconditions body

The preconditions body contains the conditions that define whether a procedure can start. They may be any combination of the following:

- wait until a given absolute time;
- wait for a given time interval to elapse;
- wait until a given condition becomes true;
- if the result of a logical expression (Boolean condition) is true;
- request a decision from the user.

A.1.4 Procedure main body

The main body of a procedure consists of a sequence of statements of the following type:

- set procedure context statement;
- initiate in parallel;
- initiate and confirm step;
- initiate and confirm activity;
- initiate activity;
- inform user statement;
- log statement.

The "initiate in parallel" statement enables a combination of steps and activities to be executed in parallel with the subsequent behaviour being one of the following:

1. the set of parallel steps or activities ends when any one of the steps or activities has ended;
2. the set ends when all of the steps or activities have ended.

A.1.5 Procedure watchdog body

The watchdog body is composed of watchdog steps. Each watchdog step monitors for the occurrence of a particular contingency condition and executes corrective actions if the condition occurs.

The purpose of the procedure watchdog body is the following:

- Detection of, and reaction to, system-level events, for example space system anomalies. The procedure watchdog body need not rectify all anomalies internally; it can suspend or abort the procedure whilst another procedure handles the anomaly.
- Ensuring that procedure-level invariant conditions are sustained for the duration of the procedure.
- Management of errors (e.g. failures) in procedure execution that cannot be handled by the watchdog of the relevant procedure main body step (see A.1.7.5).

  NOTE Error handling following execution of a procedure is managed by the calling entity.

Contingency situations monitored for by concurrently active watchdog steps are independent of each other. The reason for this is that when a watchdog step is triggered, it suspends the main body of the corresponding procedure whilst it takes the appropriate recovery action, but any other watchdog steps that are active continue their monitoring. Consequently, if there is any potential interaction between two or more contingency actions, the design of the procedure ensures that they are handled within the same watchdog step.

A.1.6 Procedure confirmation body

Within the confirmation body, a procedure can be confirmed by any combination of the following conditions:

- wait until a given absolute time;
- wait for a given time interval to elapse;

- wait until a given condition becomes true;
- if the result of a logical expression (Boolean condition) is true;
- request a decision from the user.

A.1.7 Structure of a step

A.1.7.1 Step definition

The structure of a step is identical to the structure of a procedure: a step is composed of an optional declaration body, a preconditions body (optional for a main body step, mandatory for a watchdog step), a mandatory main body, an optional watchdog body and an optional confirmation body.

The declarations at step-level (see A.1.7.2) are different from those at procedure-level and the set of statements that can be used in a step main body (see A.1.7.4) is different from that which can be used in a procedure main body.

A step has a name, which is unique within a procedure. This name may be used to refer to the step.

Whilst steps within a procedure main body may have a watchdog body, watchdog steps do not have a watchdog body.

The preconditions body is mandatory for a watchdog step (since the preconditions body defines the contingency condition for which the watchdog step is monitoring).

A.1.7.2 Step declaration body

The following are declared at step level:

- Local variables: the objects defined for internal use within a step.

  Local variables are similar to parameters in their definition and use. They have an engineering value and a validity status, which is "**invalid**" until the variable is first assigned a value.

  Local variables can be used by steps contained within a step and also by watchdog steps within the step watchdog body.

- Local events: the events that can be raised within the step and which are only accessible from within that step.

A.1.7.3 Step preconditions body

Within the preconditions body of a main body step, the conditions that define whether a step can start may be any combination of the following:

- wait until a given absolute time;
- wait for a given time interval to elapse;
- wait until a given condition becomes true;
- if the result of a logical expression (Boolean condition) is true;
- request a decision from the user.

However, the preconditions body of a watchdog step consists of a single “wait” condition.

A.1.7.4 Step main body

The main body of a step consists of a sequence of statements of the following type:

- set step context statement;
- assignment statement;
- flow control statements:
  - if statement;
  - case statement;
  - loop statements:
    - while statement;
    - for statement;
    - repeat statement;
- wait statement;
- object operation request statement;
- save context statement;
- initiate in parallel (steps or activities or both);
- initiate and confirm step;
- initiate and confirm activity;
- initiate activity;
- inform user statement;
- log statement.

A.1.7.5 Step watchdog body

A step watchdog body manages the following:

- detection of, and reaction to, any failures that occur within the main body of the corresponding step;
- ensuring that step-level invariant conditions are sustained for the duration of the step.

NOTE Error handling following execution of a step is managed by the calling entity.

A.1.7.6 Step confirmation body

Within the confirmation body, a step can be confirmed by any combination of the following conditions:

- wait until a given absolute time;
- wait for a given time interval to elapse;
- wait until a given condition becomes true;
- if the result of a logical expression (Boolean condition) is true;
- request a decision from the user.

A.2 The behaviour of a procedure

A.2.1 Procedure execution flow

Procedures can be initiated by different mechanisms within the EMCS, e.g. by a user via an MMI or automatically as a call from another procedure.

During the execution of a procedure, its “execution status” changes to reflect the path that it follows ("**preconditions**", "**executing**", "**confirmation**" or "**completed**").

On completion, the “confirmation status” of a procedure changes to reflect its success or failure ("**confirmed**", "**not confirmed**" or "**aborted**"). Prior to completion, the confirmation status of the procedure is "**not available**".

The execution and confirmation statuses can be used by the calling entity to monitor and control the execution of the procedure.

a. When the procedure is initiated, the procedure preconditions body is executed (execution status: "**preconditions**", confirmation status: "**not available**"). The outcome is one of the following:

   - the preconditions are satisfied, resulting in the initiation of the watchdog body (i.e. the initiation of all watchdog step preconditions bodies) and then the initiation of the main body;
   - the preconditions are not satisfied, resulting in the aborting of the procedure (execution status: "**completed**", confirmation status: "**aborted**").

   If the procedure has no user-defined preconditions, the watchdog body and main body are started immediately.

b. The procedure main body consists of a sequence of statements, which are of the type “initiate” (initiate activity, initiate and confirm step, initiate and confirm activity or initiate steps or activities in parallel), or are informative in nature (inform user or log), or define the context in which statements are executed.

   When the main body is started, the first statement in the sequence is executed. Subsequent statements are executed according to the sequence logic (execution status: "**executing**", confirmation status: "**not available**").

   The initiate and confirm statements (step or activity) include a continuation test, which determines how the main body continues based on the final value of the confirmation status. The outcome is one of the following:

   - continue the execution of the main body;
   - restart the current step or activity, i.e. restart from the preconditions body (if defined);
   - raise a local event that is caught by a watchdog step;
   - abort the procedure (execution status: "**completed**", confirmation status: "**aborted**").

c. The watchdog body comprises a number of (watchdog) initiate and confirm step statements. When the watchdog body is started, all of its step statements are initiated in parallel. Each watchdog step is designed to detect a particular contingency situation.

- If none of these contingency situations arises during the procedure execution, the watchdog body is automatically terminated when the procedure main body ends execution;
- If a contingency arises, the main body of the procedure is suspended whilst the contingency is handled by the relevant watchdog step (execution status: "**executing**", confirmation status: "**not available**").

  Suspension of the main body means that the currently executing atomic statement [1,2] is completed, but the subsequent action (if any) indicated as a result of that statement is not taken.

  The subsequent flow depends on the success or otherwise of the recovery and the "continuation action" arising within the watchdog step. The following cases can arise:
  - The contingency is rectified and control is returned to the procedure main body (continuation action: "**resume**", execution status: "**executing**", confirmation status: "**not available**"). The execution of the main body is resumed at the end of the previously executing atomic statement. The watchdog step that handled the contingency is re-started.
  - The contingency is handled by the watchdog by achieving the procedure goal through an alternative path. The procedure main body and the procedure watchdog body are then terminated and the procedure confirmation body is started (continuation action: "**terminate**", execution status: "**confirmation**", confirmation status: "**not available**").
  - The contingency cannot be handled by this watchdog step, in which case either a local event is raised, which triggers another watchdog step (continuation action: "**raise event**"), or the procedure is aborted (continuation action: "**abort**", execution status: "**completed**", confirmation status: "**aborted**").
- If a second watchdog step is triggered whilst the first is still executing, the two watchdog steps are executed in parallel. Control is only returned to the procedure main body when both watchdog steps have completed execution. The watchdog steps are restarted as soon as they complete execution. If either watchdog step yields an "**abort**" continuation action, the procedure is aborted immediately.

d. When the main body has completed execution (i.e. when all initiated steps and activities have completed execution [3]), the watchdog body is terminated and the confirmation body is executed (execution status: "**confirmation**", confirmation status: "**not available**").

e. When the confirmation body has completed execution, the procedure execution is completed. The outcome is one of the following:
   - the confirmation conditions are fulfilled (execution status: "**completed**", confirmation status: "**confirmed**");

---

1 An atomic statement is the lowest level of statement within the procedure language. All statements except "initiate and confirm step", "initiate activity", "initiate and confirm activity" and "initiate in parallel" are atomic in nature.

2 In the case that the main body is executing an "initiate in parallel" statement, all executing statements are suspended when all currently executing atomic statements are completed.

3 This includes any parallel steps and activities that were initiated as part of an "initiate in parallel" statement with an "**until one completes**" termination condition.

- the confirmation conditions are not fulfilled (execution status: "**completed**", confirmation status: "**not confirmed**").

If the procedure has no user-defined confirmation conditions, an implicit condition is used instead that depends on whether or not the confirmation body has been initiated by a watchdog step terminate action:

- If the confirmation body has been initiated by a “terminate” action of a watchdog step, the procedure confirmation status is set to the confirmation status of the watchdog step.
- Otherwise, the confirmation status is set to:
  - "**confirmed**" if all the activities and steps that have been initiated in the main body have been confirmed.
  - "**not confirmed**" in all other cases.

Figure A-2 shows the different execution states through which a procedure can pass when it is executed and the allowed transitions between these states.

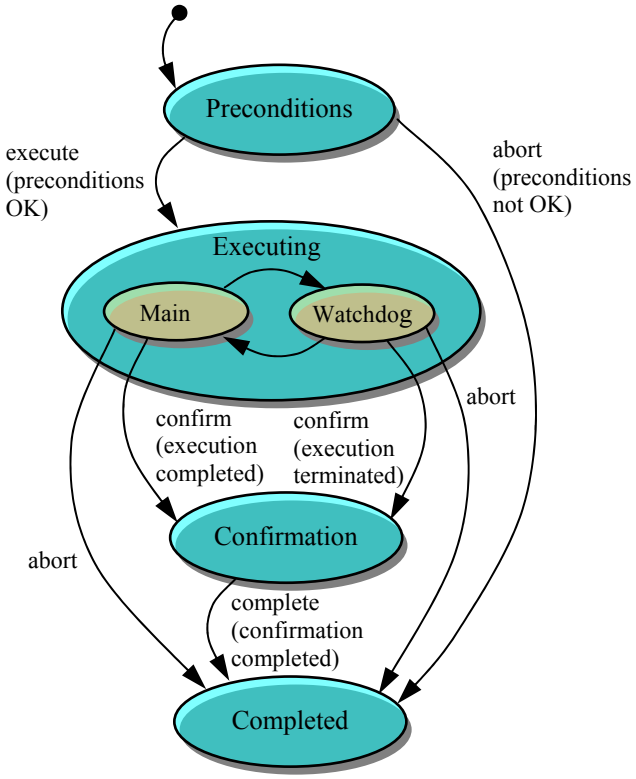


Figure A-2: Execution states and transitions for a procedure

A.2.2 Step execution flow

Steps are initiated by the execution of one of the following statements:

- within a procedure (or step) main body: “initiate and confirm step” or “initiate in parallel”;
- within a procedure (or step) watchdog body: “initiate and confirm step”.

The initiate statement implies both the execution of the step bodies and the "continuation action" derived from the execution result which determines the subsequent course of action.

a. The execution of the step preconditions body and main body proceeds in the same manner as described in A.2.1 for the procedure preconditions and main bodies.

b. The step watchdog body comprises a number of watchdog steps, each designed to detect a particular contingency:

- If none of these contingency situations arises during the step execution, the step watchdog body is automatically terminated when the step main body ends execution.
- If a contingency arises, the main body of the step is suspended whilst the contingency is handled by the relevant watchdog step (execution status: "**executing**", confirmation status: "**not available**").

  The subsequent flow depends on the success or otherwise of the recovery and the continuation action arising within the watchdog step. The following cases can arise:
  - The contingency is rectified and control is returned to the step main body (continuation action: "**resume**", execution status: "**executing**", confirmation status: "**not available**"). The watchdog step that handled the contingency is re-started.
  - The contingency is handled by the watchdog by achieving the step goal through an alternative path. The step main body and the step watchdog body are then terminated and the step confirmation body is started (continuation action: "**terminate**", execution status: "**confirmation**", confirmation status: "**not available**").
  - The contingency cannot be handled by this watchdog step, in which case either a local event is raised, which triggers another watchdog step (continuation action: "**raise event**") or the step is aborted (continuation action: "**abort**", execution status: "**completed**", confirmation status: "**aborted**").
- If two watchdog steps are triggered at the same time, their subsequent behaviour is the same as described for a procedure.

c. When the step main body has completed execution, the step watchdog body is terminated and the execution of the step confirmation body is initiated (execution status: "**confirmation**", confirmation status: "**not available**").

d. When the confirmation body has completed execution, the step execution is completed (confirmation status: "**confirmed**" or "**not confirmed**").

e. For a step, the decision on how to proceed following execution is defined within the continuation test, depending on the value of the step confirmation status ("confirmed", "**not confirmed**" or "**aborted**"). A continuation action is defined for each confirmation status, either explicitly within the initiate step statement, or by default. This is unlike for a procedure, where the decision on how to proceed following execution remains with the calling entity.

If the outcome of the continuation test is "restart", the step restarts (execution status: "preconditions"). Otherwise, the step completes execution (execution status: "completed").

If there are no user-defined preconditions or confirmation conditions for a given step, default conditions are used. These default conditions are the same as defined in A.2.1 for a procedure.

Figure A-3 shows the different execution states through which a step can pass when it is executed and the allowed transitions between these states.

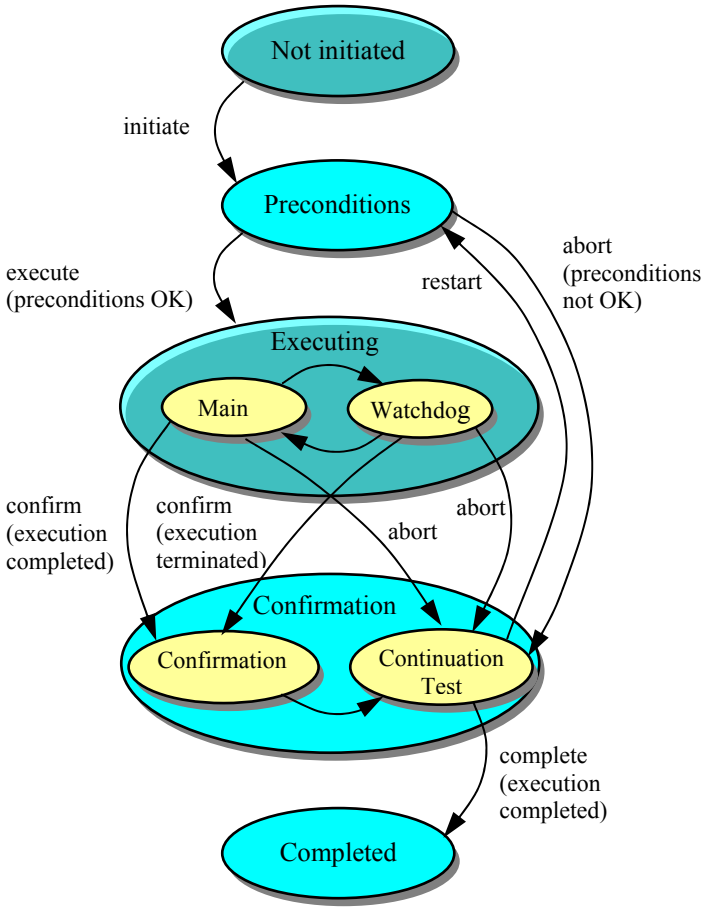


Figure A-3: Execution states and transitions for a step

A.2.3 Activity execution flow

Activities are initiated directly within a procedure or within a step by the execution of either an “initiate activity” or an “initiate and confirm activity” statement. The control of the execution of an activity is managed by the EMCS.

Figure A-4 shows the different executions states through which an activity can pass when it is executed and the allowed transitions between these states. Depending on how the activity is implemented, e.g. as a telecommand or a procedure, different notifications are reported by the EMCS for the monitoring of the progress of execution by the initiating entity. These notifications correspond to the state transitions of Figure A-4, but also to sub-state transitions which are specific to different activity types (e.g. for telecommand, routing stages such as “released from EMCS”, “reception by ground station” and “reception on-board”).

a. In the case of an “initiate activity” statement, the calling entity (procedure or step) proceeds with the execution of the next statement without waiting for notification of completion of the activity.

b. In the case of an “initiate and confirm activity”, the decision on how to proceed following execution is defined within the continuation test, depending on the value of the confirmation status ("**confirmed**", "**not confirmed**" or "**aborted**"). As for a step, the continuation action is defined for each confirmation status, either explicitly within the initiate statement, or by default.

If the outcome of the continuation test is "**restart**", the activity restarts (execution status: "**preconditions**"). Otherwise, the activity completes execution (execution status: "**completed**").

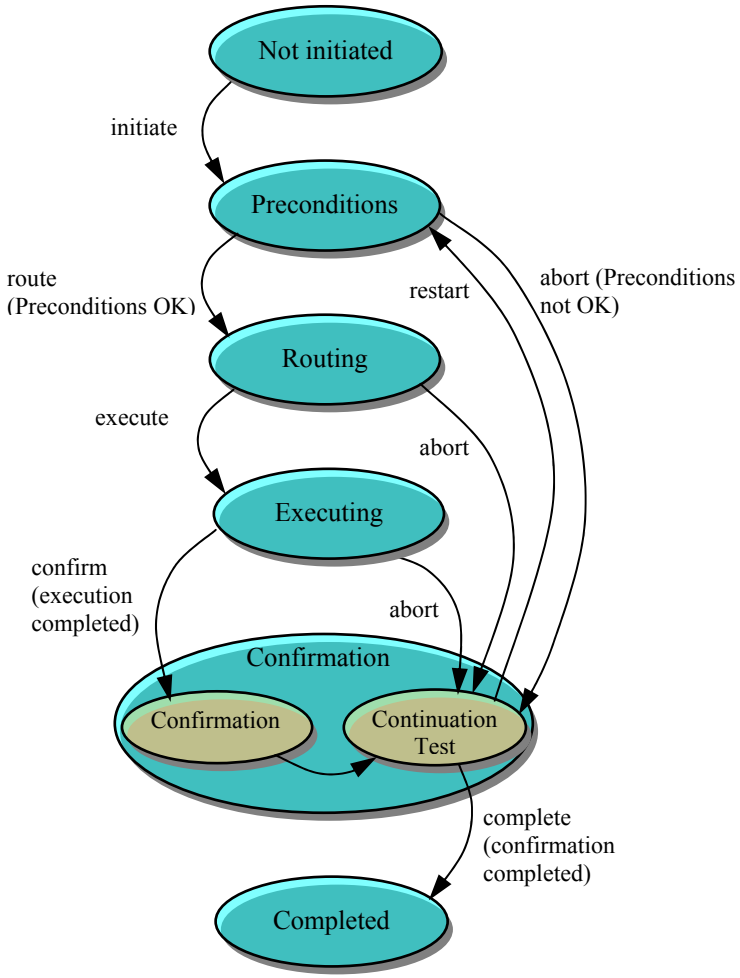


Figure A-4: Execution states and transitions for an activity

A.2.4 Execution in parallel

When an “initiate in parallel” statement is executed, all constituent steps and activities are initiated in parallel.

a. Where the termination condition of the “initiate in parallel” statement is defined as "**until one completes**", the first step or activity to complete initiates the execution of the next statement in the main body of the

calling entity (procedure or step). All other parallel steps and activities continue to execute after the first one has completed. When these other parallel steps and activities complete, the continuation actions indicated in their respective continuation tests are taken, except if the specified action is "**continue**" when no further action is taken.

b. Where the termination condition of the "initiate in parallel" statement is defined as "**until all complete**", the continuation actions evaluated within the individual steps and activities (or input by the user, if "**ask user**" is the continuation action) can be different.

- When the action returned by a given step or activity is "**abort**", the procedure (or step) is immediately aborted (including aborting any other parallel steps and activities that are still executing).
- When the action returned by a given step or activity is "**restart**", the action is taken immediately and the initiated step or activity is restarted. Its execution status is reset to "**preconditions**" and its confirmation status is reset to "**not available**". The "initiate in parallel" statement cannot complete until the restarted step or activity has completed.
- When the action returned by a given step or activity is "**raise event**", the event is raised immediately, the event is then caught by a watchdog step which suspends its corresponding main body.

A.2.5 Continuation following an "initiate and confirm" statement

An "initiate and confirm" statement can have an explicit "continuation test" as part of its definition. This continuation test consists of up to three couplets of confirmation status and corresponding continuation action (one couplet for each confirmation status). In the event that no continuation test is specified (or less than three couplets are specified), default continuation actions are defined.

Figure A-5 presents the default, allowed and forbidden combinations [4] of confirmation status and continuation action for "initiate and confirm" statements that are part of a main body.

Figure A-6 presents the default, allowed and forbidden combinations of confirmation status and continuation action for "initiate and confirm" statements that are part of a watchdog.

[4] The allowed combinations of confirmation status and continuation action can be extended for special testing scenarios. For example, a test can be designed to validate the rejection of a critical command, such as "deploy solar array", where the successful outcome of the test is that the corresponding activity is "**aborted**". In this case, the continuation action "**continue**" can be defined for the confirmation status "**aborted**".

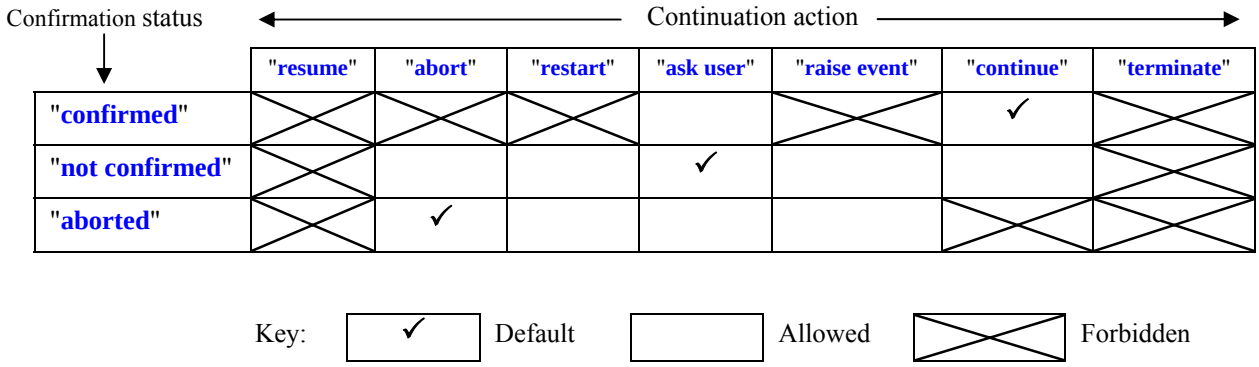


Figure A-5: Confirmation status and continuation action combinations for main body "initiate and confirm" statements

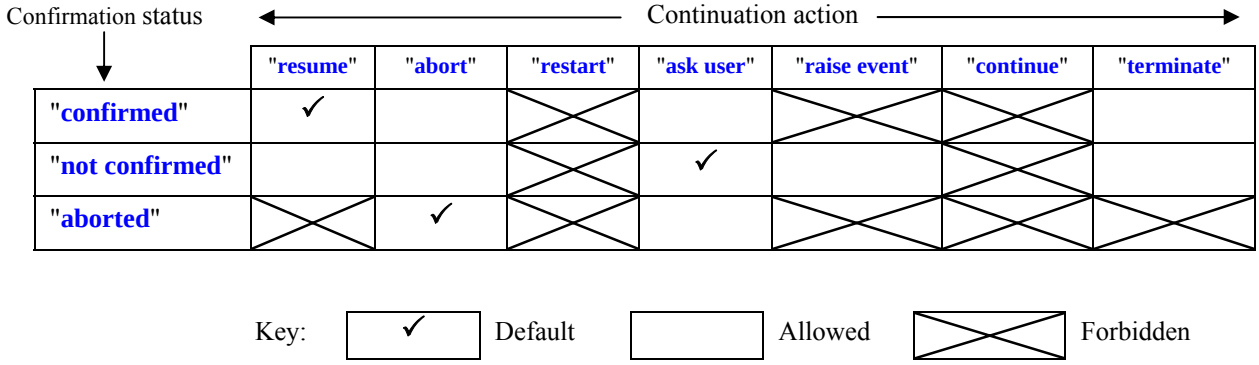


Figure A-6: Confirmation status and continuation action combinations for watchdog "initiate and confirm" statements

The meaning of the different continuation actions is as follows:

- "**resume**"

  Only used within the continuation test of a watchdog "initiate and confirm" statement. It resumes the execution of the main body of the procedure (or step) at the place at which it was suspended when the contingency was detected.

- "**abort**"

  Aborts the procedure (or step).

- "**restart**"

  Only used within the continuation test of a main body "initiate and confirm" statement and it restarts the corresponding "initiate and confirm" statement. To avoid a potentially indefinite loop, a timeout or a maximum number of iterations can be specified. The timeout is expressed as a relative time from the start of execution time of the statement. If the timeout or the maximum number of iterations is exceeded then either:

- an event is raised which terminates the statement and is then handled by a corresponding watchdog step, or
- the parent procedure (or step) is aborted.

- "**ask user**"

  Asks the user to specify how to proceed.

  For a main body "initiate and confirm" statement, the user may select any continuation action except "**resume**", "**ask user**" and "**terminate**". For a watchdog "initiate and confirm" statement, the user may select any continuation action except "**continue**" and "**ask user**".

- "**raise event**"

  Raises a local event, which is caught by a watchdog step of the procedure (or step). When the watchdog step has performed its function, the execution of the main body of the procedure (or step) resumes at the place at which it was suspended when the watchdog was triggered.

- "**continue**"

  Only used within a continuation test of a main body "initiate and confirm" statement. It implies that the procedure (or step) continues to be executed according to the content of the main body.

- "**terminate**"

  Only used within the continuation test of a watchdog "initiate and confirm" statement. It prematurely ends the execution of the procedure (or step) main body and starts the execution of the confirmation body.

In the case of a watchdog "initiate and confirm step" statement, the step itself is re-initiated unless the continuation action is "**abort**".

A.3 PLUTO language definition

A.3.1 Conventions

The terminology used in defining the syntax of the procedure language is compatible with ISO/IEC 14977.

The syntax of the procedure language is a set of rules, collectively known as a grammar.

Each rule defines a construct of the language, known as a "non-terminal symbol". A non-terminal symbol is a combination of zero or more non-terminal symbols and "terminal symbols". A terminal symbol is a sequence of one or more characters forming an irreducible element of the language.

A terminal symbol may consist of one or more words separated by one or more separators. The space, tab and end-of-line characters are separators.

The definition of the language constructs uses a graphical convention known as a "railroad" diagram, an example of which is shown in Figure A-7. The elements of this graphical convention are listed below:

- the name at the top of the diagram is that of the non-terminal symbol being defined;
- the main line corresponds to mandatory elements of the construct;
- branch lines correspond to optional elements;
- "return" lines correspond to optional repetitions of one or more elements;

- non-terminal symbols are enclosed in rectangles;
- terminal symbols are enclosed in round-cornered boxes (in the limit, a circle).

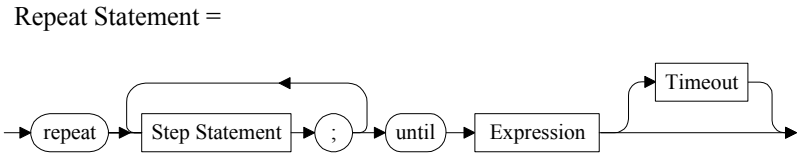


Figure A-7: Example railroad diagram

Figure A-7 defines that a repeat statement starts with the keyword “repeat” followed by one or more step statements, each separated by a “;”, followed by the keyword “until” and an expression (whose result, when true, terminates the repeat loop), followed by an optional timeout.

A.3.2 Language case sensitivity

Only engineering units (as defined in Annex B) are case sensitive. All other words used in the language are not case-sensitive.

A.3.3 Comments

Comments may be inserted within a procedure. Comments have no effect on the execution of a procedure.

Comments begin and end with the character pair symbols "**/***" and "***/**" respectively.

A.3.4 Keywords

A number of words have a special meaning in the procedure language. If any of these words are used in the naming of space system objects (e.g. in the name of a parameter), it is important to ensure that there is no potential ambiguity for end-users when interpreting a procedure. For the system implementing the procedure language, the ambiguity is resolved by the application of precedence rules that are either specified implicitly by the grammar or defined explicitly in this Standard.

These words are called “keywords” and are shown in bold face and between straight quotes.

EXAMPLE "**procedure**", "**not confirmed**", "**>=**", "**XOR**", "**execution status**", "**acos**", "**is contained in**"

NOTE 1 Spaces (such as white space, tab and carriage return) within keywords are treated as single blanks.

NOTE 2 Keywords are not case-sensitive.

A.3.5 Identifiers

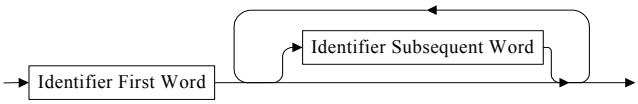
An identifier denotes a name and consists of one or more words separated by spaces, where each word is a sequence of letters and digits.

An identifier is unique within its own context. For example, step identifiers are unique within a procedure, but the same step identifier may be used in several procedures.

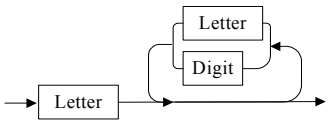
Reference to an object outside its context is achieved by use of an "object reference path" (see A.3.9.8 for the "Object Reference" language construct).

An identifier has the general form:

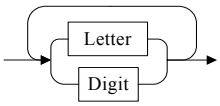
Identifier =



Identifier First Word =



Identifier Subsequent Word =



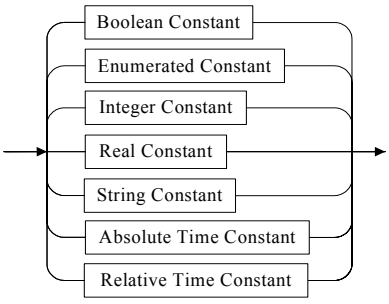
where:

- Letter is an upper-case or lower-case letter of the alphabet;
- Digit is one of the decimal characters from 0 to 9.

A.3.6 Constants

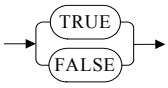
A constant can be one of the following:

Constant =



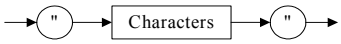
a. A Boolean constant is represented as:

Boolean Constant =



b. An enumerated constant is represented by:

Enumerated Constant =



where Characters is any sequence of letters or digits or one of the following characters:

space ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~

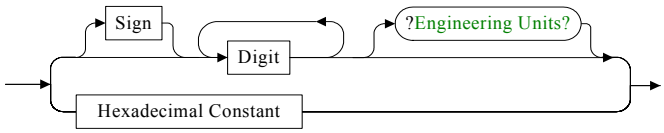
To enter a double quote, it is preceded by a reverse solidus character (i.e. "\").

To enter a reverse solidus character, it is preceded by a reverse solidus character.

EXAMPLE "red", "yellow", "green", "not confirmed"

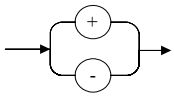
c. An integer constant is represented as:

Integer Constant =



where Sign is as follows:

Sign =



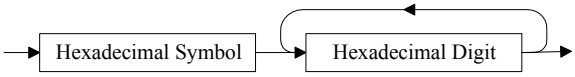
Digit is one of the decimal characters from 0 to 9.

?Engineering Units? is one of the engineering units defined in Annex B.

NOTE The "?" is a special-sequence-symbol which indicates the start and end of a special sequence (see ISO/IEC 14977); in this case a standalone syntax for Engineering Units.

Hexadecimal Constant is defined as follows:

Hexadecimal Constant =

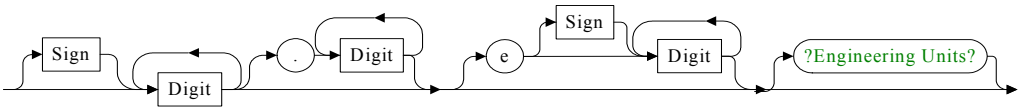


- The Hexadecimal Symbol is the character pair symbol "**0x**".
- The Hexadecimal Digit is a decimal character from 0 to 9 or a letter from A to F;

EXAMPLE 23 **A**, 2056, **0x**2056, **0x**FFFF

d. A real constant is represented as:

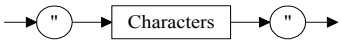
Real Constant =



EXAMPLES 123.56e12, 0.1, 23E6, 5.3 **[kg/m^3]**

e. A string constant is represented as:

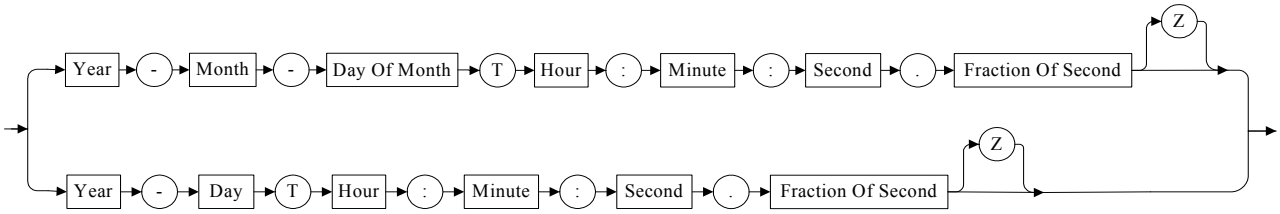
String Constant =



EXAMPLE "The double-quote character is : \". "

f. An absolute time constant is represented as:

Absolute Time Constant =



1. Month/day of month calendar variation

   Year-Month-Day Of MonthTHour:Minute:Second.Fraction Of Second(Z)

   where:

   Year = four digits with a value in the range 0001-9999;

   Month = two digits with a value in the range 01-12;

   Day Of Month = two digits with a value in the range 01-28, 01-29, 01-30, or 01-31;

   "T" = calendar-time separator;

   Hour = two digits with a value in the range 00-23;

   Minute = two digits with a value in the range 00-59;

   Second = two digits with a value in the range 00-59 (00-58 or 00-60 during leap seconds);

   Fraction Of Second = one to n digits. To obtain a given precision, the appropriate number of digits to the right of the period is used;

   "Z" = optional terminator.

EXAMPLE 2001-08-18T21:07:43.137468Z

2. Year/day of year calendar variation

   Year-DayTHour:Minute:Second.Fraction Of Second(Z)

   where:

Year, "T", Hour, Minute, Second, Fraction Of Second, "Z": are as defined in 1. above;

Day = three digits with a value in the range 001-365 or 001-366.

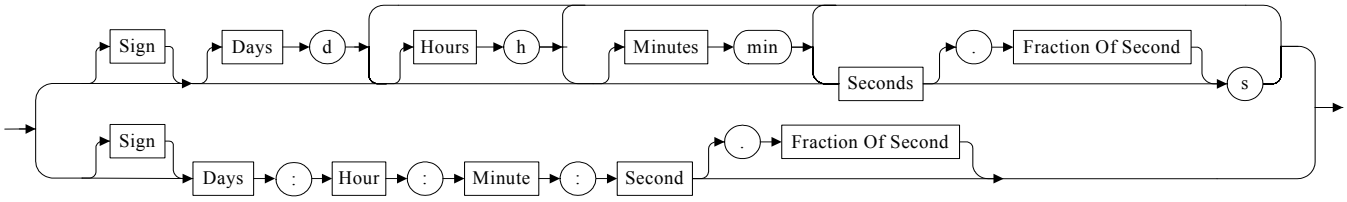
EXAMPLE 2001-033T13:21:32.226

NOTE 1 Leading zeros are included to make up the specified number of digits.

NOTE 2 Elements shown in brackets are optional.

g. A relative time (duration) constant is represented as:

Relative Time Constant =



where:

Days, Hours, Minutes and Seconds are unsigned integers;

Hour, Minute, Second and Fraction Of Second are as defined in f. above.

EXAMPLES 30 **h** 10 **min**, 200 **s**, - 0:00:10:00

A.3.7 Types

A type defines the set of values that variables and expressions possessing that type may assume.

The type of a variable determines its use in various contexts. For example, in an assignment, it is mandatory that the left and right-hand sides of the assignment are type-compatible.

The procedure language supports the predefined types shown in Table A-1.

Table A-1: Predefined types

Data type	Definition
Boolean	Defines the set of truth values denoted by the predefined constants true and false
Enumerated set reference	Defines a reference to a set of enumerated values[a]
Signed integer	Defines an implementation subset of signed integer values
Unsigned integer	Defines an implementation subset of unsigned integer values
Real	Defines an implementation subset of real values
String	Defines a set of values that are character strings
Absolute time	Defines a set of values that represent an absolute date such as 2003-02-15
Relative time	Defines a set of values that represent an interval of time such as 2 **s**
Property value set	Defines an enumerated set comprising all values of a property of a system element, reporting data, activity or event Example 1 "application process identifier of system element" returns the enumerated set consisting of all defined APIDs. Example 2: "name of parameter of power subsystem" returns the enumerated set consisting of all names of parameters belonging to the power subsystem.
Property data type	Defines a data type inherited from a property of a given system element, reporting data, activity or event This can result in one of the following: • A simple value type, e.g. "value of Voltage of Battery1" returns the type "real with engineering units **V"**; • A list of value type, e.g. "value of status of my network printer" returns the enumerated set {"ready", "paused", "printing", "unable to connect"}.
[a] Enumerated sets can be defined on a system-wide basis or locally within a procedure and are referenced by name, e.g. the enumerated set named Event Type comprises the set of values {"Normal", "Low Severity", "Medium Severity", "High Severity"}.	

A.3.8 System interfaces

The SSM objects relevant to the procedure language are introduced in subclause 4.2.2. Each type and subtype of SSM object has a number of properties and operations that are accessible using services provided by the EMCS.

"Initiate" and "Initiate and Confirm" are two important EMCS services for activities which can be invoked using dedicated statements of the procedure language. In addition, the language provides two generic mechanisms to invoke other services of the EMCS:

- The Object Property Request which is used to return a property of an object (e.g. get the initiation time of an activity, get the status of a printer). The minimum sets of "standard" property requests for activities, reporting data and events are specified in A.3.9.36.
- The Object Operation Request Statement which is used to perform an operation on an object (e.g. clear printer queue, open/close operating system file). The minimum sets of "standard" operation requests for activities and reporting data are specified in A.3.9.24.

"Standard" means that these services are supported by all EMCS implementations. A given implementation of the procedure language may also support non-standard services.

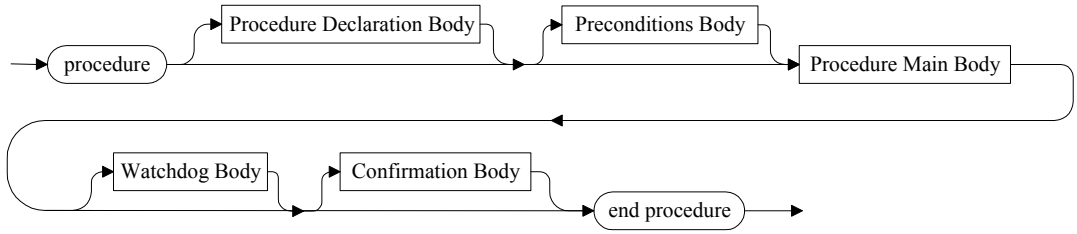
A.3.9 Language constructs

A.3.9.1 Procedure Definition

Meaning — Defines the elements of a procedure.

Syntax

Procedure Definition =



Definition

Procedure Declaration Body

Declares the local events that can be raised within the procedure.

Preconditions Body

Defines the preconditions of the procedure.

Whilst the procedure is waiting for the initial conditions to be satisfied, its execution status is "**preconditions**".

Procedure Main Body

Contains the statements that achieve the goal of the procedure.

Whilst the main body is executing, the execution status is "**executing**".

Watchdog Body

Handles contingency situations defined for the procedure.

Whilst the watchdog body is handling a contingency situation, the execution status is "**executing**".

Confirmation Body

Defines the conditions under which the procedure is confirmed.

Whilst the procedure is waiting to be confirmed, its execution status is "**confirmation**".

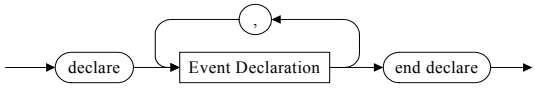
A.3.9.2 Procedure Declaration Body

Meaning

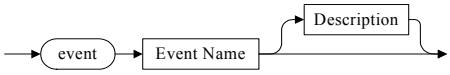
Declares the local events that can be raised within a procedure.

Syntax

Procedure Declaration Body =

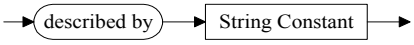


Event Declaration =



Event Name =

Description =



Definition

Event Declaration

Declares a local event that can be raised within the procedure.

Event Name

The name of the local event.

Local events that are declared at procedure level are associated with the procedure and are only used by the procedure watchdog body (e.g. they are not used within steps of the procedure main body).

Description

The (optional) description of the local event.

Example

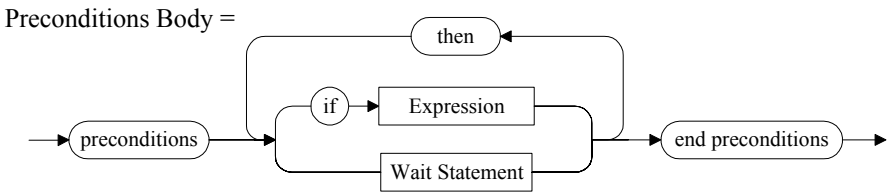
Procedure Name	Slew Manoeuvre
PLUTO script	**procedure** **declare** **event** Manoeuvre Damping Failed **described by** "Cross-track error not reducing to less than 5 arcmin within 100 s of manoeuvre completion" **end declare** <……> **end procedure**

A.3.9.3 Preconditions Body

Meaning

Defines the conditions under which the execution of a procedure (or step) can proceed once it is initiated.

Syntax



Definition

Expression

Yields a result which, if true, allows the associated procedure (or step) to be executed.

Wait Statement

Defines one of the following:

- the absolute time when the execution of the procedure (or step) can start;
- an interval of time after which execution can start;
- a condition to be true or an event to occur before execution can start.

The "**then**" loop is used to specify a combination of different conditions.

If the expression is not satisfied following evaluation or the optional timeout in the wait condition is reached, then the procedure (or step) is skipped completely and the procedure (or step) confirmation status is set to "**aborted**".

The preconditions body of a watchdog step is always present and consists of a single “wait statement”.

Example

Procedure Name	Switch on Gyro5 in Fine Mode

PLUTO script	**procedure**
	  preconditions
	  wait until Gyro Temperature > 60 **degC**
	  end preconditions
	  main
	  initiate and confirm Switch on Gyro Converter**;**
	  initiate and confirm Switch on Gyro5**;**
	  initiate and confirm Gyro5 Fine Mode**;**
	  end main
	end procedure

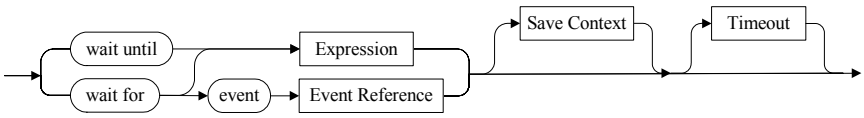
A.3.9.4 Wait Statement

Meaning

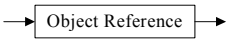
Defines a delay.

Syntax

Wait Statement =



Event Reference =



Definition

Expression

Yields an absolute time or a Boolean result (wait until) or a relative time (wait for).

An absolute time causes the calling entity (procedure or step) to be delayed until the given time.

A Boolean condition causes the calling entity to be delayed until the condition is satisfied.

A relative time delays the calling entity by the given time interval.

Event Reference

The reference to a local event or a system-level event associated with a system element, activity or reporting data (an object reference of type event).

Save Context

Creates a local copy (a snapshot) of one or more SSM reporting data.

This language construct is included within a wait statement to ensure that the snapshot of SSM reporting data pertains to the instant at which the condition is satisfied or the event occurs.

Timeout

Defines a relative time to be applied as a timeout for the wait statement.

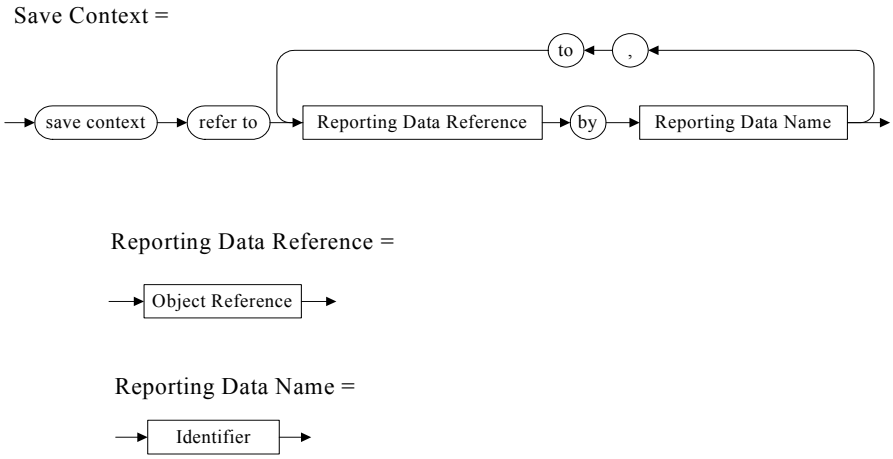
The timeout is not available where the wait condition is either “for a relative time” or “until an absolute time”.

When the wait statement appears in a preconditions body, the raise event option (following a timeout) is not available, since the corresponding watchdog body has not yet been started. Therefore, the procedure (or step) is aborted if the specified timeout interval is exceeded.

A.3.9.5 Save Context

Meaning — Creates a copy of reporting data for local use.

Syntax



Definition — Reporting Data Reference

The reporting data reference (an object reference of type reporting data) from which the local reporting data is copied.

Reporting Data Name

The name of the local reporting data.

Reporting data has a life cycle that is independent of the procedure execution environment. Reporting data can therefore be updated during the execution of a set of procedure statements.

The purpose of this language construct is to create a snapshot copy of one or more reporting data. The construct is only intended for use where more than one property of a given reporting data is used at different times within a procedure and the contemporaneity of those properties is important.

Local reporting data is associated with the procedure or step in which it is defined. Prior to creation (i.e. saving the context), its validity status is "**not available**". During creation, local reporting data acquires the properties of the reporting data from which it is copied.

The life cycle and “visibility” of local reporting data is limited to the current instance of the procedure within which it is created, i.e. it is not persistent between successive executions of the procedure.

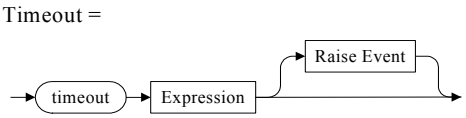
Example

Procedure Name	Eclipse Operations
PLUTO script	**procedure** **main** <……> **wait for event** Eclipse Entry **save context refer** **to** Temperature **of** Battery1 **by** TempBatt1**,** **to** Voltage **of** Battery1 **by** VoltBatt1**;** **<……>** **end main** **end procedure**

A.3.9.6 Timeout

Meaning — Defines a timeout interval for the current statement.

Syntax



Definition — Expression

Yields a relative time from the statement start of execution time and gives the timeout interval after which, if the statement has not completed execution, either:

- an event is raised which terminates the statement and is then handled by a corresponding watchdog step, or
- the parent procedure (or step) is aborted.

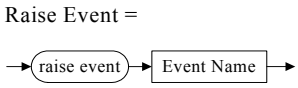
Raise Event

Raises a local event.

A.3.9.7 Raise Event

Meaning — Raises a local event.

Syntax



Definition — Event Name

The name of the local event.

Example

Procedure Name	Slew Manoeuvre
PLUTO script	**procedure** **declare** **event** Manoeuvre Damping Failed**,** <……> **end declare** **main** <……> **wait for** Crosstrack Error **<** 5 **arcmin** **timeout** Manoeuvre End Time **+** 100 **s** **raise event** Manoeuvre Damping Failed**;** **<……>** **end main** **watchdog** **initiate and confirm step** Recover Damping Failure **preconditions** **wait for event** Manoeuvre Damping Failed **end preconditions** <……> **end step;** **end watchdog** **end procedure**

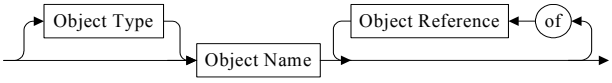
A.3.9.8 Object Reference

Meaning

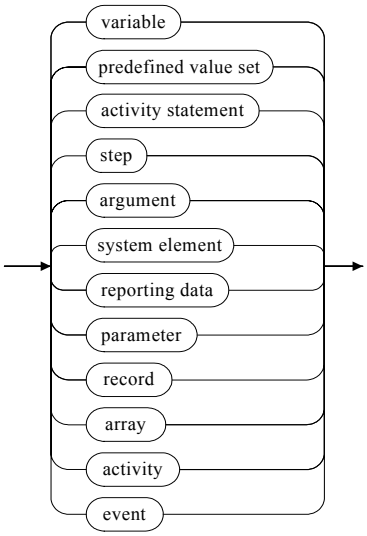
Refers to an object by means of a reference path.

Syntax

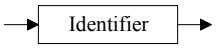
Object Reference =



Object Type =



Object Name =



Definition

Object Reference

The reference path for the object.

Object Type

The explicit type of the object.

Object Name

The name of the object, which may be an activity, an object within an activity (e.g. a step, a variable, a procedure argument), a given initiation instance of an activity (i.e. an activity statement), reporting data (i.e. a parameter, a compound parameter or a component of a compound parameter), an event or a system element.

A component of a compound parameter can be a simple component (i.e. a parameter), an array or a record.

Where the object type is not given explicitly and the result of the object reference is not unique within the overall SSM, the reference is resolved according to the following precedence rules (1 = highest precedence, 14 = lowest precedence):

1. variable
2. local reporting data

3. local event
4. activity statement
5. step
6. argument
7. system element
8. reporting data
9. parameter
10. record
11. array
12. activity
13. predefined value set
14. system event

Example

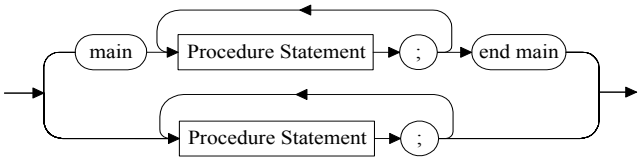
PLUTO script	Temperature **of** Catalyst Bed **of** Thruster X **of** Thruster Main Branch

A.3.9.9 Procedure Main Body

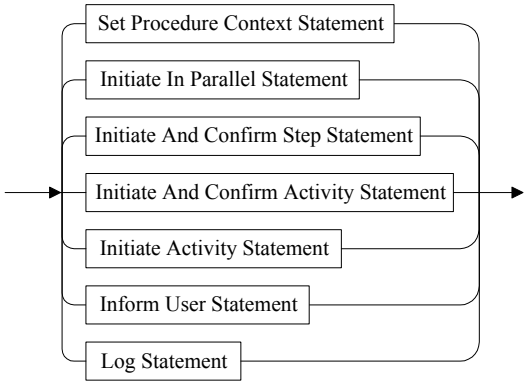
Meaning — Contains the statements that achieve the goal of the procedure.

Syntax

Procedure Main Body =



Procedure Statement =



Definition — Procedure Statement

A statement allowed within a procedure main body, which is one of the following:

a. Set Procedure Context Statement

   Defines the context in which a set of procedure statements executes (overriding the current default context).

b. Initiate In Parallel Statement

   Provides the capability to execute steps or activities (or both) concurrently. Each step or activity within the parallel statement has an execution path of its own. The parallel step finishes when one step or activity completes execution or they all complete execution.

c. Initiate And Confirm Step Statement

   Initiates and confirms the execution of a step.

d. Initiate And Confirm Activity Statement

   Initiates and confirms the execution of an activity.

e. Initiate Activity Statement

   Initiates the execution of an activity. The initiated activity proceeds in parallel with the initiating procedure.

f. Inform User Statement

   Provides the capability to output a message for acknowledgement by the user.

g. Log Statement

Provides the capability to log a message to the procedure execution log.

Example

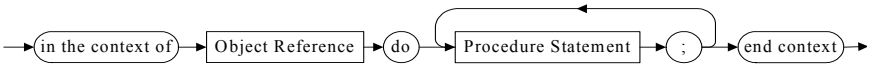
Procedure Name	Switch on Gyro5 in Fine Mode
PLUTO script	**procedure** **initiate and confirm step** Switch on Gyro5 Converter **main** **initiate and confirm** Switch on Gyro Converter**;** **end main** **end step;** **initiate and confirm step** Power on Gyro5 **main** **initiate and confirm** Switch on Gyro5**;** **initiate and confirm** Gyro5 Fine Mode**;** **end main** **end step;** **end procedure**

A.3.9.10 Set Procedure Context Statement

Meaning — Defines the context in which a set of procedure statements executes.

Syntax

Set Procedure Context Statement =



Definition — This statement explicitly defines the context in which a set of procedure statements executes. This overrides the current default context which is the one in which the procedure is currently executing.

Object Reference
: This is either a system element or reporting data (either a compound parameter or a record-type component of a compound parameter).

Procedure Statement
: A statement allowed within a procedure main body.

Example

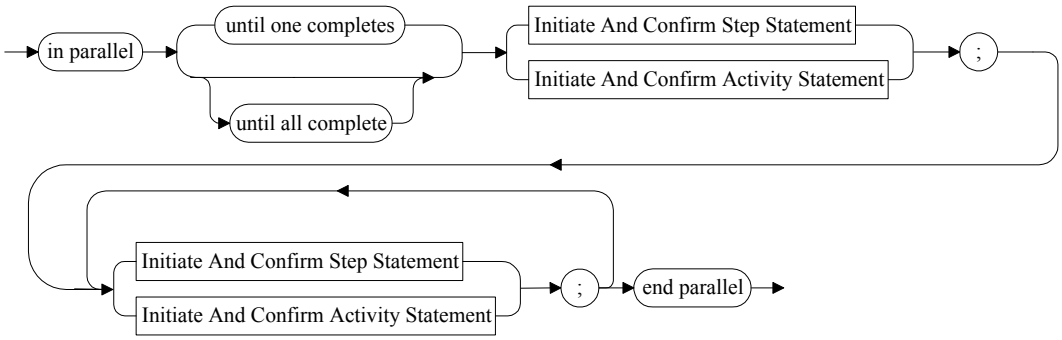
Procedure Name	Test telescopes
PLUTO script	**procedure** <.....> **in the context of** Telescope1 **do** **initiate and confirm** Power on**;** **initiate and confirm** Take image**;** **initiate and confirm** Process and display image**;** **initiate and confirm** Power off**;** **end context;** **in the context of** Telescope2 **do** **initiate and confirm** Power on**;** **initiate and confirm** Take image**;** **initiate and confirm** Process and display image**;** **initiate and confirm** Power off**;** **end context;** <.....> **end procedure**

A.3.9.11 Initiate In Parallel Statement

Meaning — Defines a set of steps or activities (or both) that are executed in parallel.

Syntax

Initiate In Parallel Statement =



Definition

Initiate And Confirm Step Statement

Initiates and confirms the execution of a step.

Initiate And Confirm Activity Statement

Initiates and confirms the execution of an activity.

When the parallel statement is executed, each of its constituent steps or activities is initiated in parallel. The termination of the execution depends on the selected termination condition, as defined below:

- "**until all complete**" (default)

  Terminates execution if all of the constituent steps or activities are ended.

- "**until one completes**"

  Terminates execution if any of its constituent steps or activities is ended.

If neither of these conditions is explicitly specified, the default behaviour is "**until all complete**".

Example

Procedure Name	Switch on Gyro3 and Gyro5 in Fine Mode
PLUTO script	**procedure** **preconditions** **wait until** Gyro3 and Gyro5 Converter = "ON" **end preconditions** **main** **in parallel until all complete** **initiate and confirm step** Switch on Gyro3 in Fine Mode **preconditions** **wait until** Temperature **of** Gyro3 > 60 **degC** **end preconditions** **main** **initiate and confirm** Switch on Gyro3**;**

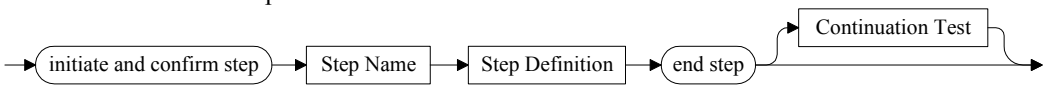
	initiate and confirm Gyro3 Fine Mode**;** **end main** **end step;** **initiate and confirm step** Switch on Gyro5 in Fine Mode **preconditions** **wait until** Temperature **of** Gyro5 **>** 60 **degC** **end preconditions** **main** **initiate and confirm** Switch on Gyro5**;** **initiate and confirm** Gyro5 Fine Mode**;** **end main** **end step;** **end parallel;** **end main** **end procedure**

A.3.9.12 Initiate And Confirm Step Statement

Meaning — Initiates a designated step and waits for confirmation of its execution.

Syntax

Initiate and Confirm Step Statement =



Step Name =

Definition

Step Name

The name of the step.

The identifier is unique within the procedure.

Step Definition

Defines the step.

Continuation Test

Defines how the execution of the initiating procedure (or step) proceeds after the step has been executed. Default actions exist in the case that no continuation test is specified (see A.3.9.33 on Continuation Test for more details on the default actions).

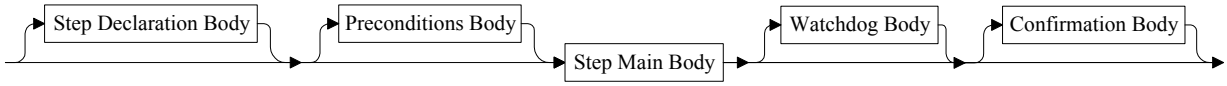
The initiated step finishes completely before the initiating procedure (or step) may proceed.

A.3.9.13 Step Definition

Meaning — Defines the elements of a step.

Syntax

Step Definition =



Definition

Step Declaration Body

Declares the local objects of the step.

Preconditions Body

Defines the preconditions of the step.

Whilst the step is waiting for the initial conditions to be satisfied, its execution status is "**preconditions**".

Step Main Body

Contains the statements that achieve the sub-goal of the step.

Whilst the main body is executing, the execution status of the step is "**executing**".

Watchdog Body

Handles contingency situations defined for the step.

Whilst the watchdog body is handling a contingency situation, the execution status of the step is "**executing**".

Confirmation Body

Defines the conditions under which the step is confirmed.

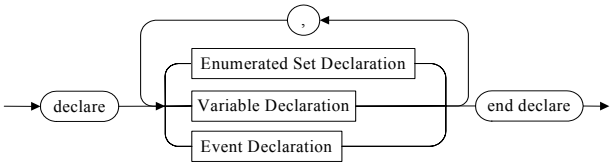
Whilst the step is waiting to be confirmed, its execution status is "**confirmation**".

A.3.9.14 Step Declaration Body

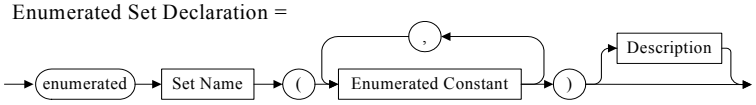
Meaning — Declares the local variables used by a step and the local events that can be raised within it.

Syntax

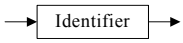
Step Declaration Body =



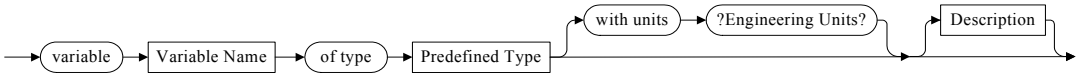
Enumerated Set Declaration =



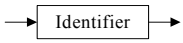
Set Name =



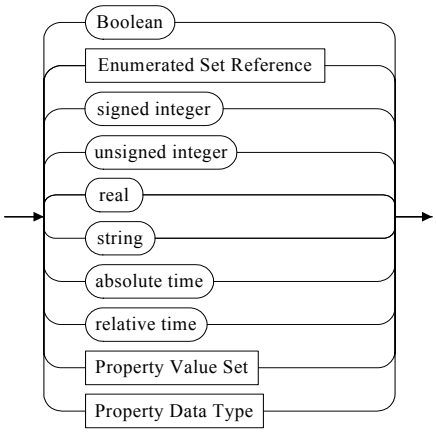
Variable Declaration =



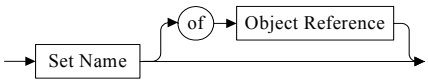
Variable Name =



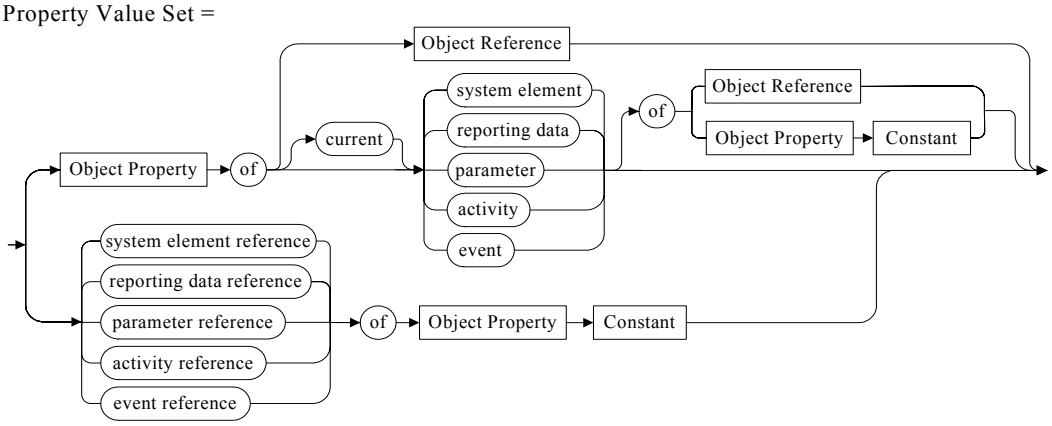
Predefined Type =



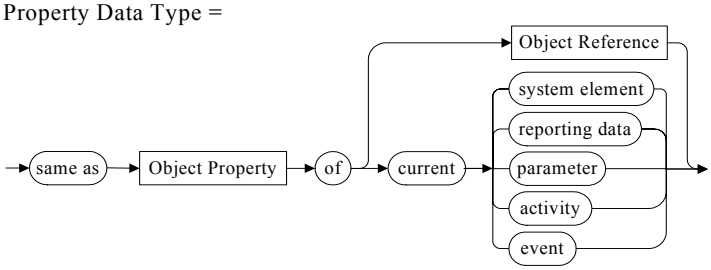
Enumerated Set Reference =



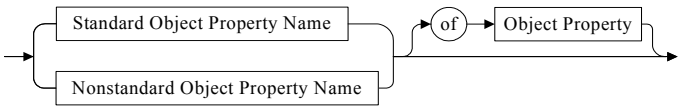
Property Value Set =



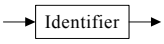
Property Data Type =



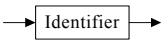
Object Property =



Standard Object Property Name =



Nonstandard Object Property Name =



Definition

Enumerated Set Declaration

Declares a set of enumerated values that is used locally within the step.

Set Name

The name of the enumerated set.

Description

The (optional) description of the enumerated set.

Variable Declaration

Declares a local variable that is used within the step.

A local variable has a validity status, sampling time and value. It is associated with the step in which it is declared and its scope is limited to the step and its lower level steps. Prior to its first assignment, its validity status is "**not available**".

Variable Name

The name of the local variable.

Predefined Type

The data type of the local variable.

Enumerated Set Reference

The reference to the enumerated set (and the SSM object to which it is attached).

Property Value Set

An enumerated set comprising all values of a property of a system element, reporting data, activity or event.

Property Data Type

A data type of a property of a given system element, reporting data, activity or event.

Object Property

This can be either a standard property of an activity, step, reporting data, variable, procedure argument or event, or a non-standard object property.

Standard Object Property Name

The name of the standard object property.

Nonstandard Object Property Name

The name of the non-standard object property.

Description

The (optional) description of the local variable.

Event Declaration

Declares a local event that can be raised within the step.

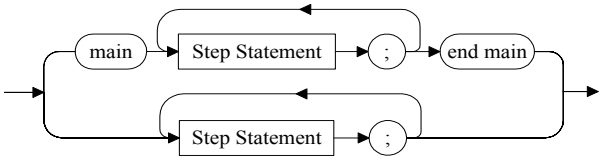
Local events that are declared at step level are associated with the step and can only be used by the step watchdog body (e.g. they cannot be used within the parent procedure or step or lower-level steps).

A.3.9.15 Step Main Body

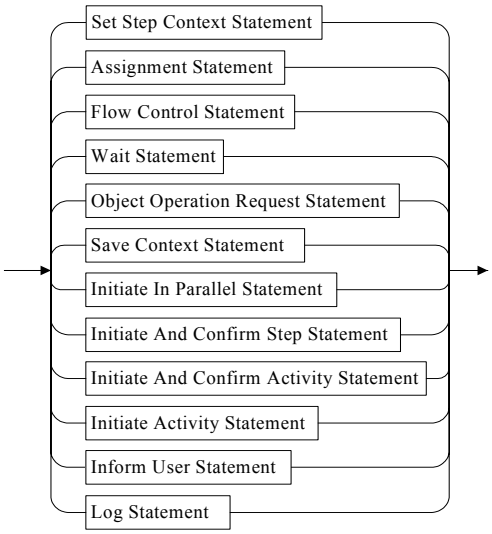
Meaning — Contains the statements that achieve the sub-goal of the step.

Syntax

Step Main Body =



Step Statement =



Definition — Step Statement

A statement allowed within a step main body, which is one of the following:

a. Set Step Context Statement

   Defines the context in which a set of step statements executes (overriding the current default context).

b. Assignment Statement

   Replaces the current value of a variable with another computed value.

c. Flow Control Statement

   This group of statements controls the execution flow through the step. There are of two types of statement: conditional or repetitive statements.

d. Wait Statement

   Causes the delay of a step by a time interval, until a given absolute time, until the condition is true or until an event occurs.

e. Object Operation Request Statement

   Requests a specified operation to be performed on an SSM object, namely a system element, activity, reporting data, event

or step.

f. Save Context Statement

   Makes a copy of one or more SSM reporting data for local use within the step to ensure that all properties used are simultaneously sampled.

g. Initiate In Parallel Statement

   Provides the capability to execute steps or activities concurrently. Each step or activity within the parallel statement has an execution path of its own. The parallel step finishes when one step or activity completes execution or they all complete execution.

h. Initiate And Confirm Step Statement

   Initiates and confirms the execution of a step.

i. Initiate And Confirm Activity Statement

   Initiates and confirms the execution of an activity.

j. Initiate Activity Statement

   Initiates the execution of an activity. The initiated activity proceeds in parallel with the initiating step.

k. Inform User Statement

   Provides the capability to output a message for acknowledgement by the user.

l. Log Statement

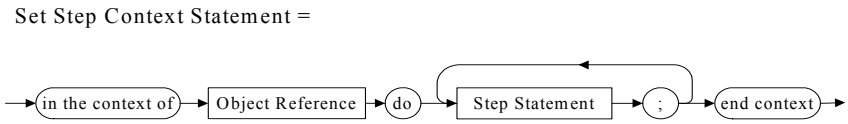
   Provides the capability to log a message to the procedure execution log.

A.3.9.16 Set Step Context Statement

Meaning

Defines the context in which a set of step statements executes.

Syntax



Definition

This statement explicitly defines the context in which a set of step statements executes. This overrides the current default context which is the one in which the step is currently executing.

Object Reference
This is either a system element or reporting data (either a compound parameter or a record-type component of a compound parameter).

Step Statement
A statement allowed within a step main body.

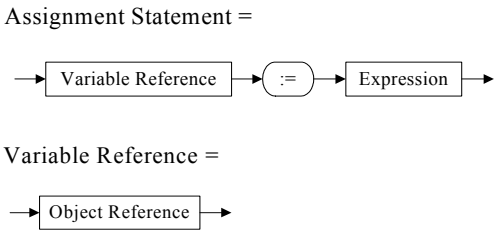
Example

Procedure Name	Payload switch on
PLUTO script	**procedure** <.....> **initiate and confirm step** Switch on Telescopes **in the context of** Telescope1 **do** **initiate and confirm** Power on**;** **wait for** 5 **s;** **initiate and confirm** Switch on HT**;** **wait for** 5 **s;** **initiate and confirm** Set HT **with** Voltage **:=** 2000 **V** **end with;** **end context;** **in the context of** Telescope2 **do** **initiate and confirm** Power on**;** **wait for** 5 **s;** **initiate and confirm** Switch on HT**;** **wait for** 5 **s;** **initiate and confirm** Set HT **with** Voltage **:=** 2000 **V** **end with;** **end context;** **end step;** <.....> **end procedure**

A.3.9.17 Assignment Statement

Meaning — Assigns a value to a local variable.

Syntax



Definition — Variable Reference

The reference to the local variable that receives the value of the expression (an object reference of type variable).

A step only uses a variable which has been declared in the list of local variables of the step.

The visibility of a variable is limited to the step and its lower-level steps. Where a variable name is reused within a lower-level step, the variable of the parent step is no longer accessible.

Expression

A combination of mathematical and language elements that is processed at run-time to compute the value of the variable.

Example

PLUTO script	Voltage **:=** Value **of** T001 ***** 10

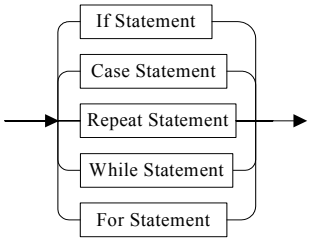
A.3.9.18 Flow Control Statement

Meaning

Conditionally or iteratively executes a set of statements.

Syntax

Flow Control Statement =



Definition

If Statement

Executes one list of statements or another depending on the result of a logical condition.

Case Statement

Executes one of several lists of statements depending on a logical condition.

Repeat Statement

A conditional iteration statement, where the condition is evaluated at the end.

While Statement

A conditional iteration statement, where the condition is evaluated at the beginning.

For Statement

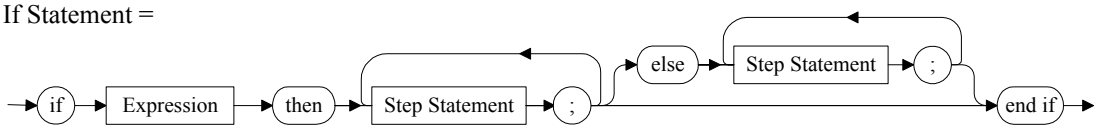
An iteration statement, where the number of iterations is fixed (but evaluated at run-time).

A.3.9.19 If Statement

Meaning — Executes one statement (or set of statements) or another depending on the result of a logical condition.

Syntax

If Statement =



Definition

Expression

The expression returns a logical result which, if true, results in the execution of the first statement (or set of statements) and, if false, results in the execution of the second statement (or set of statements).

Step Statement

A statement allowed within a step main body.

Example

Procedure Name	Slew Manoeuvre

PLUTO script	**procedure**
	<……>
	if Target Offset < 0.05 **deg then**
	initiate and confirm Go to Fine Pointing**;**
	else
	initiate and confirm Centre Target**;**
	initiate and confirm Go to Fine Pointing**;**
	end if;
	<……>
	end procedure

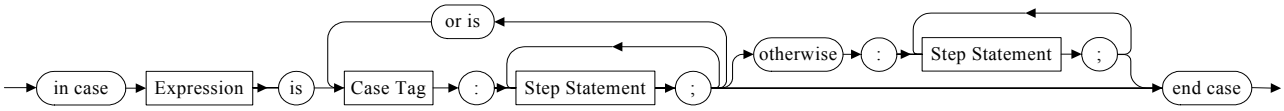
A.3.9.20 Case Statement

Meaning

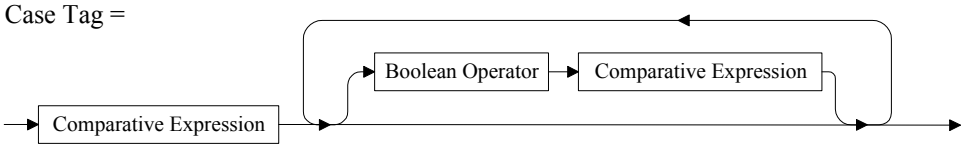
Multiple conditional branching depending on the value of an expression.

Syntax

Case Statement =



Case Tag =



Definition

Expression

A combination of mathematical and language elements which returns a value.

Case Tag

A comparative expression which, when combined with the "**in case**" expression, determines whether the corresponding step statement (or set of statements) is executed.

The type of the case tags all correspond to the type of the "**in case**" expression.

Step Statement

A statement allowed within a step main body.

Boolean Operator

One of the following operators: "**AND**", "**OR**" or "**XOR**" (see also Expression).

Relational Operator

One of the following operators: "**=**", "**!=**", "**<**", "**>**", "**<=**" or "**>=**" (see also Expression).

The case statement works in the following way:

1. If the value of the "**in case**" expression corresponds to the value of one of the case tags, then the corresponding statement (or set of statements) is executed.
2. If the value of the "**in case**" expression does not correspond to any of the values of the case tags, and if there is an "**otherwise**" case tag, then the corresponding statement (or set of statements), and only that one, is executed.
3. Otherwise, no statement (or set of statements) is executed.

If two or more case tags have the same value, only the first one is processed.

If the "**in case**" expression corresponds to the value of two or more case tags, only the first one is processed.

Example

PLUTO script	**in case** Temperature **of** Biolab Experiment **is <** 5 **degC :** Switch On Heater1**;** Switch On Heater2**;** **or is <** 10 **degC :** Switch On Heater1**;** **otherwise:** **inform user** "The temperature of Biolab is \"Nominal\", no action taken"**;** **end case**

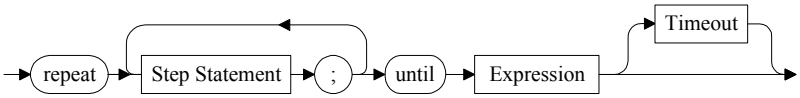
A.3.9.21 Repeat Statement

Meaning

Conditional iteration of a statement (or set of statements) with control at the end, optionally terminated by a timeout.

Syntax

Repeat Statement =



Definition

Step Statement

A statement allowed within a step main body.

Expression

Yields a logical result which, when true, terminates the iteration.

Timeout

Defines an (optional) timeout interval after which, if the preceding expression is not true, either:

- an event is raised which terminates the statement and is then handled by a corresponding watchdog step,

or

- the parent step is aborted.

The repeat statement differs from the while statement in that the condition is evaluated after each execution of the statement (or set of statements).

As a result, the statement (or set of statements) is executed at least once.

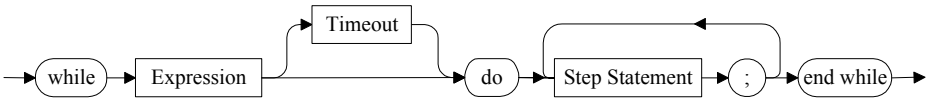
A.3.9.22 While Statement

Meaning

Conditional iteration of a statement (or set of statements) with control at the beginning, optionally terminated by a timeout.

Syntax

While Statement =



Definition

Expression

Yields a logical result which, if false, terminates the iteration.

Timeout

Defines an (optional) timeout interval after which, if the preceding expression is still true, either:

- an event is raised which terminates the statement and is then handled by a corresponding watchdog step,

or

- the parent step is aborted.

Step Statement

A statement allowed within a step main body.

The while statement differs from the repeat statement in that the condition is evaluated before each execution of the statement (or set of statements). As a result, the statement (or set of statements) may never be executed at all.

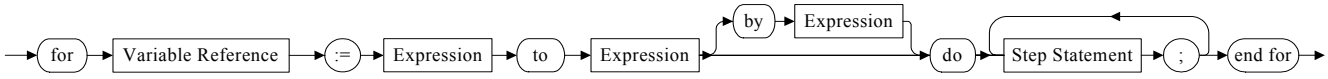
If the expression is true and there is more than one step statement in the loop, then the complete set of statements is executed, i.e. it is not interrupted if the timeout occurs during its execution; the timeout is invoked at the end.

A.3.9.23 For Statement

Meaning Iterates the execution of a statement (or set of statements) a fixed number of times.

Syntax

For Statement =



Definition

Variable Reference

The reference to a variable that is used as a counter (object reference of type variable).

"**:=**" Expression

An integer expression, which is computed at run-time to determine the first value that is assigned to the counter.

"**to**" Expression

An integer expression, which is computed at run-time to determine the last value of the counter before leaving the statement.

"**by**" Expression

An integer expression, which is computed at run-time to determine by what value the counter is increased or decreased, after each iteration of the execution of the list of statements.

When the "**by**" expression is not present, the default behaviour is "increase by 1"

Step Statement

A statement allowed within a step main body.

The value of the counter may be referred to within the list of statements, but not changed. As a result, the counter is not used as the left-hand side of an assignment statement.

All the expressions are only evaluated once at the beginning of the "**for**" loop.

Example

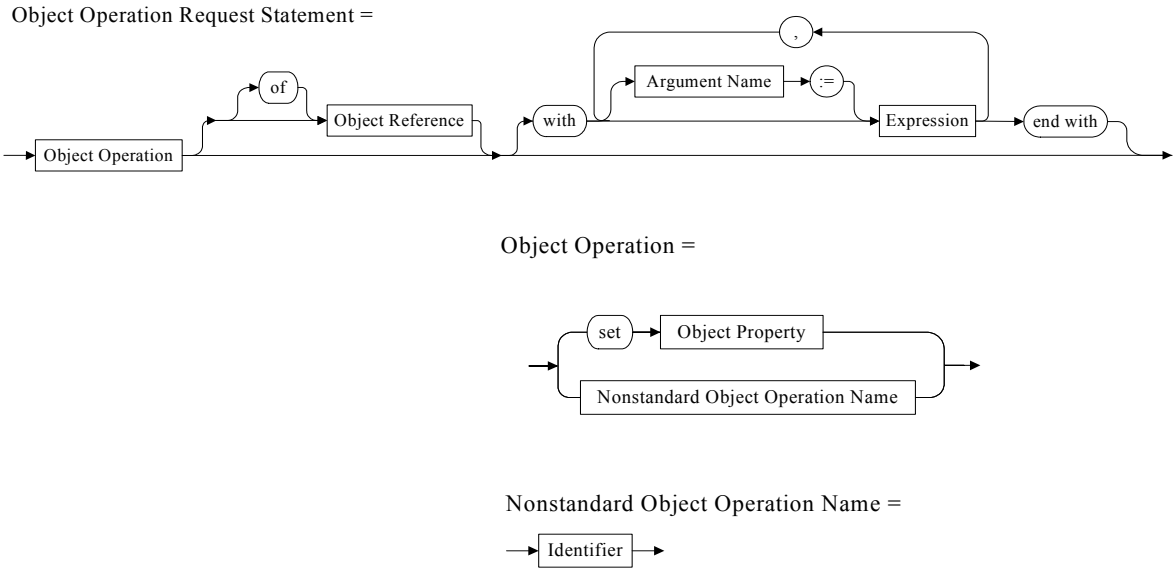
Procedure name	Enable Payload Thermal Control Lines
Arguments	Number of Heater Lines: unsigned integer

PLUTO script	**procedure**
	  preconditions
	  wait until All Payloads **=** "OFF"
	  end preconditions
	  main
	  initiate and confirm step Enabling
	  declare
	  unsigned integer Counter
	  end declare
	  main
	  for Counter **:=** 1 **to** Number of Heater Lines
	  do
	  initiate and confirm Enable Thermal Control Line
	  with Line Number **:=** Counter **end with;**
	  end for;
	  end main
	  end step;
	  end main
	end procedure

A.3.9.24 Object Operation Request Statement

Meaning Invokes an operation of an object.

Syntax



Definition

Object Operation

The object operation that is requested, which can either be an operation to set a property of an object or any other operation defined for an object within the EMCS.

Nonstandard Object Operation Name

The name of the non-standard object operation.

Argument Name

The name of the argument.

Expression

Yields a result that defines the value of the argument.

The standard operations that can be requested for an activity or step are listed in Table A-2 and for reporting data, variable or procedure argument in Table A-3 (there are no standard operations for an event).

The availability of individual operations depends on the type of the object.

Table A-2: Activity and step operation requests

Operation request	Meaning	Argument	Result
"**set confirmation status**"	Sets the confirmation status of the activity or step	"**not available**" or "**confirmed**" or "**not confirmed**" or "**aborted**"	None

Table A-3: Reporting data, variable and argument operation requests

Operation request	Meaning	Argument	Result
"**set validity status**"	Sets the validity status of a parameter, a variable or a procedure argument	"**not available**" or "**valid**" or "**invalid**"	None
"**set value**"	Sets the engineering value of a parameter, a variable or a procedure argument	any pre-defined type	None
"**set monitoring status**"	Sets the overall monitoring status of a parameter	"**not available**" or "**nominal**" or "**failed**"	None
"**set status consistency check status**"	Sets the status consistency check status of a parameter	"**not available**" or "**nominal**" or "**failed**"	None
"**set limit check status**"	Sets the limit check status of a parameter	"**not available**" or "**danger high**" or "**warning high**" or "**within limits**" or "**warning low**" or "**danger low**"	None
"**set delta check status**"	Sets the delta check status of a parameter	"**not available**" or "**nominal**" or "**failed**"	None
"**set expected check status**"	Sets the expected state check status of a parameter	"**not available**" or "**nominal**" or "**failed**"	None
NOTE The procedure language includes operations to set the value, or another property, of reporting data, variables and procedure arguments. These operations are normally performed by other functions of the space system (e.g. the spacecraft, the EMCS health monitoring function) and can only be invoked by the procedure language in configurations where these functions are not present.			

Example

PLUTO script	set value of T002 with 1.0 end with;

A.3.9.25 Save Context Statement

Meaning — Creates a copy of one or more reporting data for local use.

Syntax

Save Context Statement =

Definition — Save Context

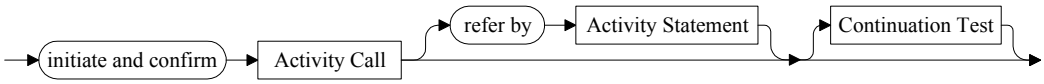
Performs the local reporting data copy.

A.3.9.26 Initiate And Confirm Activity Statement

Meaning — Initiates a designated activity and waits for confirmation of completion.

Syntax

Initiate and Confirm Activity Statement =



Activity Statement =

Definition

Activity Call

The call to the activity that is initiated.

Activity Statement

The initiation instance of the activity. The "visibility" of this activity statement is limited to the calling entity. If the initiating statement occurs within a loop, it is always the last instance of the initiating statement that is referred to.

The activity statement is a dynamic object of the EMCS of type activity. It is attached to the procedure or step in which it is defined. It inherits all operations and properties defined for activities (see A.3.9.24 and A.3.9.36) and can be included in an object reference (see A.3.9.8).

Continuation Test

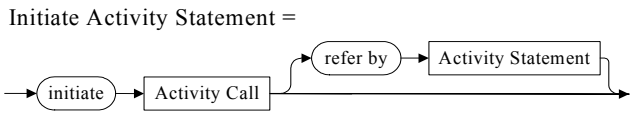
Specifies how the execution of the calling entity proceeds after the associated activity has been executed.

The initiated activity finishes completely before the initiating entity (procedure or step) proceeds.

A.3.9.27 Initiate Activity Statement

Meaning — Asynchronously initiates a designated activity.

Syntax



Definition — Activity Call

The call to the activity that is initiated.

Activity Statement

The initiation instance of the activity.

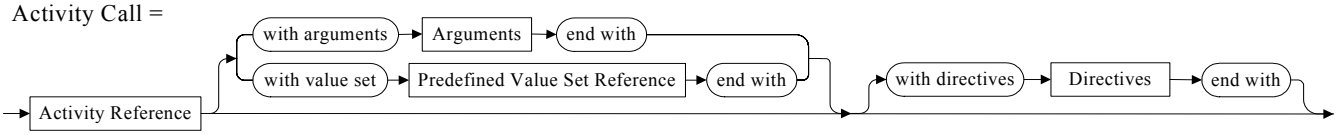
The initiated activity proceeds in parallel with the initiating entity.

A.3.9.28 Activity Call

Meaning Calls an activity.

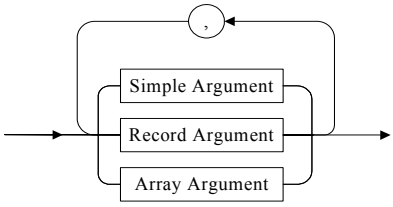
Syntax

Activity Call =



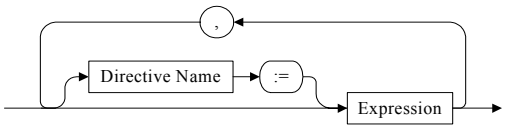
Activity Reference =

Arguments =

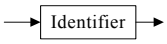


Predefined Value Set Reference =

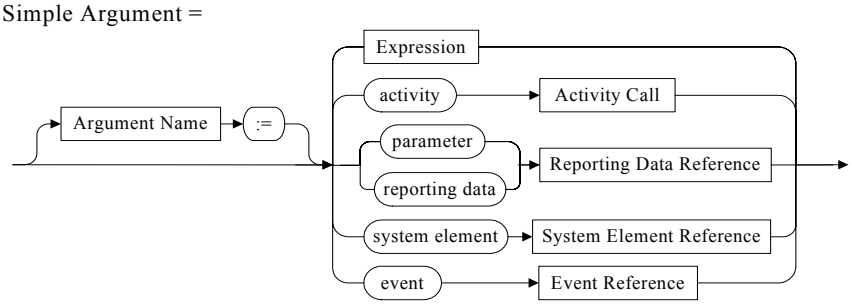
Directives =



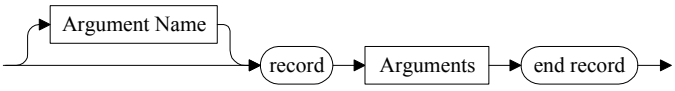
Directive Name =



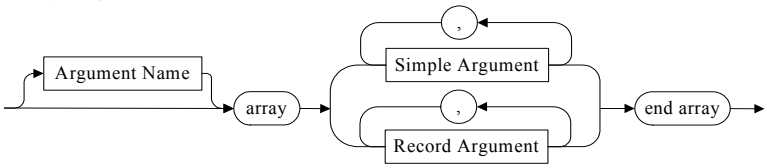
Simple Argument =



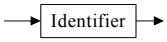
Record Argument =



Array Argument =



Argument Name =



System Element Reference =

Definition

Activity Reference

The reference to the activity (an object reference of type activity).

Arguments

The arguments of the activity. The type of the argument can be a simple argument (single value or object), an array (multiple occurrences) or a record. A record is a recursive construct of arguments of type simple or array (see also ECSS-E-70-41).

Predefined Value Set Reference

The reference to a predefined set of argument values for the activity, i.e. predefined in the operations database (an object reference of type predefined value set).

Directives

The directives used by the EMCS when initiating the activity. For example, for an activity of type telecommand, this can include:

- the MAP ID to be used;
- whether the telecommand is to be sent in AD or BD mode;
- specification of the telecommand verification packets to be generated in response to this telecommand.

Whereas arguments are specific to a given activity, directives are specific to a given type of activity (e.g. telecommands).

Directive Name

The name of the directive.

Simple Argument

An argument of simple type, i.e. a value, a parameter, an activity, a system element or an event.

Record Argument

An argument of record (of arguments) type.

Array Argument

An argument of array (of arguments) type.

Argument Name

The name of the argument.

Expression

Yields the value of the argument.

Activity Call

A call to an activity.

System Element Reference

A reference to a system element (an object reference of type system element).

Argument names are optional. If argument names are not provided, they are known implicitly from the order in which the argument values are given. Either no argument names are provided or they are all provided.

Examples

Typical examples of the use of the different types of simple argument for activities of type telecommand are:

- Reporting Data Reference: Parameter# in PUS service 12 requests;
- System Element Reference: APID in PUS service 14 requests;
- Activity Call: Inserting telecommand packets in the on-board schedule; PUS service 11,4 request.

PLUTO script	**initiate** Insert into Schedule **with array** **record** Telecommand **:= activity** Switch On Telescope**,** Release Time **:=** 2004-117T12:15:00.0Z**,** Subschedule **:=** 1**,** Interlock Set **:=** 1 **end record** **record** Telecommand **:= activity** Switch On Telescope HT**,** Release Time **:=** 2004-117T12:17:00.0Z**,** Subschedule **:=** 1**,** Interlock Set **:=** 2**,** Interlock Assessed **:=** 1 **end record** **record** Telecommand **:= activity** Set Telescope HT **with** Level **:=** 1000 **V** **end with,** Release Time **:=** 2004-117T12:19:00.0Z**,** Subschedule **:=** 1**,** Interlock Assessed **:=** 2 **end record** end array end with;

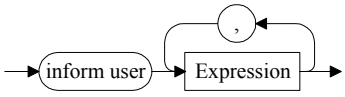
A.3.9.29 Inform User Statement

Meaning

Outputs a message for acknowledgement by the user.

Syntax

Inform User Statement =



Definition

Expression

Yields the message that is output to the user.

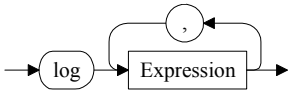
The procedure execution proceeds (up to completion) without waiting for the acknowledgement of the user.

A.3.9.30 Log Statement

Meaning — Outputs a message to the procedure execution log.

Syntax

Log Statement =



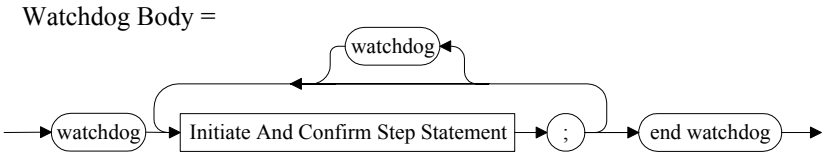
Definition — Expression

Yields the message that is output to the procedure execution log.

A.3.9.31 Watchdog Body

Meaning — Handles contingency situations defined for a procedure (or step).

Syntax



Definition — Initiate And Confirm Step Statement

An initiate and confirm statement for the steps that compose the watchdog body (called watchdog steps). When the watchdog body is executed, all of its steps are initiated in parallel. The preconditions body of a watchdog step defines the contingency situation that it is monitoring. The preconditions body of a watchdog step is always present and consists of a single "wait statement".

When more than one watchdog step is triggered, the following rules apply:

a. If any watchdog step returns the action to abort, the parent procedure (or step) aborts immediately.

b. The main body remains suspended until all watchdog steps are completed.

   1. If all actions returned are the same, the procedure (or step) proceeds as indicated.
   2. If the actions returned are different, user intervention is requested (i.e. "**ask user**").

Example

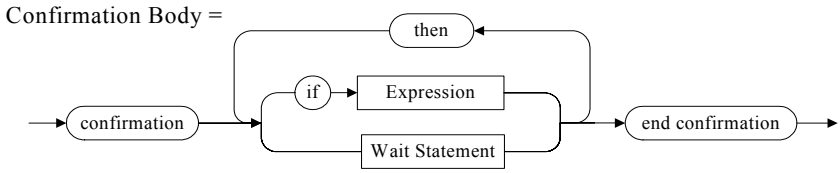
Procedure name	Data Bus Reconfiguration
PLUTO script	**procedure** **main** **initiate and confirm step** Enter Ground Intervention Mode **initiate and confirm** Activate GIM**;** **end step;** **initiate and confirm step** Reconfigure Data Bus **preconditions** **wait until** AOCS Mode **=** "GIM" **end preconditions** **initiate and confirm** Switch Bus From B To A**;** **initiate and confirm** Activate Bus Acquisition**;** **end step;** **initiate and confirm step** Exit Ground Intervention Mode **initiate and confirm** Deactivate GIM**;** **end step;** **end main**

	watchdog **initiate and confirm step** Check Depointing **preconditions** **wait until** Pitch > 10 **deg OR** Roll > 10 **deg OR** Yaw > 10 **deg** **end preconditions** **initiate and confirm** Activate Bus Acquisition**;** **initiate and confirm** Exit Ground Intervention Mode**;** **initiate and confirm** Activate Coarse Mode**;** **end step;** **end watchdog** **end procedure**

A.3.9.32 Confirmation Body

Meaning — Defines the conditions under which a procedure (or step) is confirmed.

Syntax



Definition

"if" Expression

Yields a Boolean value which, if true, sets the confirmation status of the procedure (or step) to "**confirmed**".

Wait Statement

A logical condition which, when true confirms the procedure (or step).

If the Boolean condition is not satisfied after evaluation or the (optional) timeout is reached in the wait statement, the procedure (or step) confirmation status is set to "**not confirmed**".

The "**then**" loop is used to specify a combination of different conditions.

When there is no confirmation body defined for a procedure (or step) the confirmation status is evaluated as follows:

- If all procedure-initiated (or step-initiated) steps and activities of the main body of the procedure (or step) are "**confirmed**", then the confirmation status changes from "**not available**" to "**confirmed**".
- If any of the initiated steps or activities are "**not confirmed**", then the confirmation status of the initiating procedure (or step) also becomes "**not confirmed**".
- If an initiated step or activity raises an event within its continuation test (see A.3.9.33), the confirmation status of the initiating procedure (or step) is determined by the watchdog step which caught the event.

Example

Procedure name	Switch on Gyro5
PLUTO script	**procedure** <.....> **confirmation** **wait until** Output **of** Gyro5 < 0.2 **deg/h** **end confirmation** **end procedure**

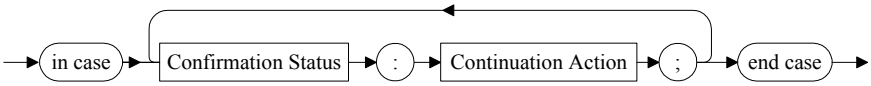
A.3.9.33 Continuation Test

Meaning

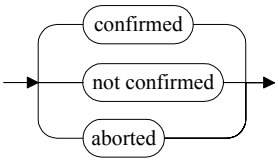
Defines how to proceed after the associated “initiate and confirm” statement is executed.

Syntax

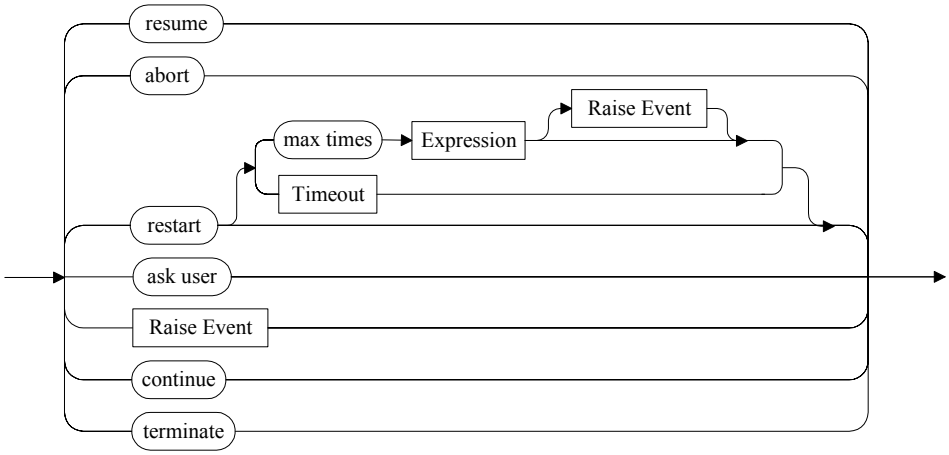
Continuation Test =



Confirmation Status =



Continuation Action =



Definition

The continuation test comprises a set of couplets of confirmation status and associated action. If a given confirmation status is not specified, a default action applies (see A.2.5).

Confirmation Status

A final state of the associated “initiate and confirm” statement, which may be one of the following:

- "**confirmed**"

  The “initiate and confirm” statement is confirmed.

- "**not confirmed**"

  The “initiate and confirm” statement is not confirmed.

- "**aborted**"

  The “initiate and confirm” statement is aborted.

Continuation Action

Defines the action to be taken when the associated “initiate and confirm” statement ends execution. It may be one of the following:

"**resume**"

Resumes the execution of the main body of the procedure (or step).

"**abort**"

Aborts the procedure (or step).

"**restart**"

Restarts the corresponding “initiate and confirm” statement. To avoid this loop continuing indefinitely, it may be terminated either by a timeout or by a specified maximum number of restarts ("**max times**").

"**ask user**"

Asks the user to specify how to proceed.

Raise Event

Raises a local event, which is caught by a watchdog step of the procedure (or step).

"**continue**"

The procedure (or step) continues execution according to the content of the main body.

"**terminate**"

Prematurely ends the execution of the procedure (or step) main body and starts the execution of the confirmation body.

Example

Procedure name	Switch on Gyro5
PLUTO script	**procedure** **main** **initiate and confirm step** Switch on Gyro5 Converter **main** **initiate and confirm** Switch on Gyro Converter **in case** **not confirmed: restart;** **aborted: ask user;** **end case;** **end main** **end step;** **initiate and confirm** Power on Gyro5 <……> **end step;** **end main** **end procedure**

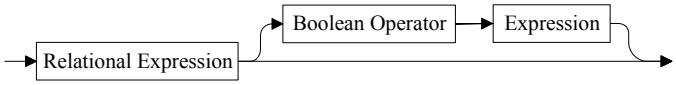
A.3.9.34 Expression

Meaning

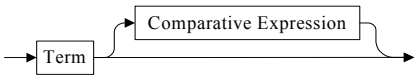
Yields a value by evaluating a grouping of constants, references and operators.

Syntax

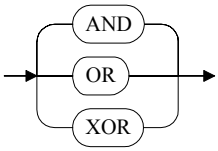
Expression =



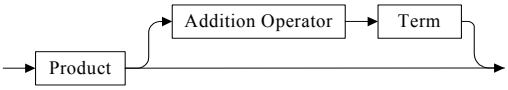
Relational Expression =



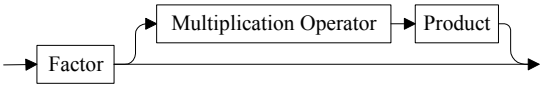
Boolean Operator =



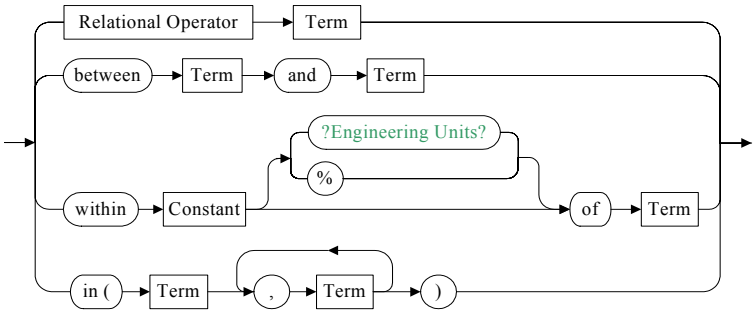
Term =



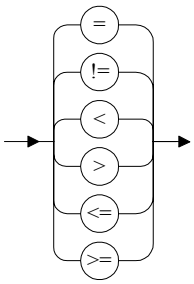
Product =



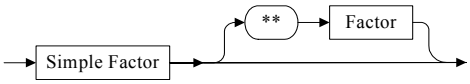
Comparative Expression =



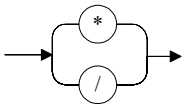
Relational Operator =



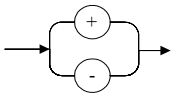
Factor =



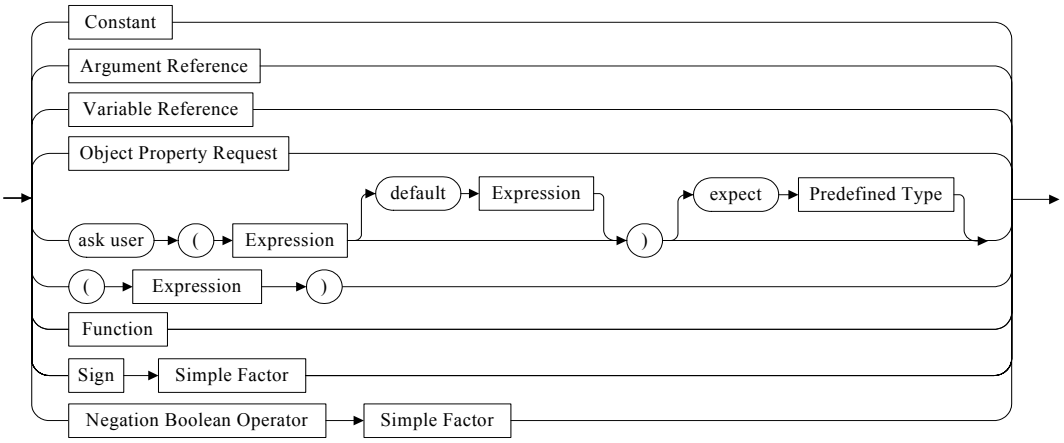
Multiplication Operator =



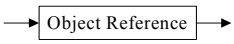
Addition Operator =



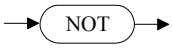
Simple Factor =



Argument Reference =



Negation Boolean Operator =



Definition

Relational Expression

Constants and references combined with relational operators.

Comparative Expression

Constants and references combined with comparative operators.

Object Property Request

A call to return a defined property of any object of the SSM (see A.3.9.36).

Function

A predefined (i.e. built-in) function of type mathematical, string or time that operates on zero or more input arguments and returns a result.

It can readily be seen that the syntax for Simple Factor contains several alternatives that can each reduce to an Identifier and hence lead to ambiguity about what is intended. The following precedence rule applies in such cases to define the order in which the possibilities are resolved (1 = highest precedence, 4 = lowest precedence):

1. local variable (searched upwards through the step hierarchy).
2. procedure argument.
3. function name.
4. object name.

The syntax of the expression expresses the precedence rules of the operators.

For each operator, Table A-4 defines the types that can be used for the operands. The operators are listed in descending order of precedence.

Table A-4: Predefined operators

Operator	Meaning	Left operand	Right operand	Result
"-"	Unary minus	integer, real	N/A	integer, real
"+"	Unary plus	integer, real	N/A	integer, real
"NOT"	Boolean NOT	Boolean	N/A	Boolean
"**"	Exponentiation	integer	integer	integer
		integer	real	real
		real	integer	real
		real	real	real
"*"	Multiplication	integer	integer	integer
		integer	real	real
		real	integer	real
		real	real	real
		relative time	integer, real	relative time
		integer, real	relative time	relative time
"/"	Division	integer	integer	real
		integer	real	real
		real	integer	real

Operator	Meaning	Left operand	Right operand	Result
		real	real	real
		relative time	integer, real	relative time
"+"	Addition	integer	integer	integer
		integer	real	real
		real	integer	real
		real	real	real
		absolute time	relative time	absolute time
		relative time	relative time	relative time
	Concatenation	string	any	string
		any	string	string
"-"	Subtraction	integer	integer	integer
		integer	real	real
		real	integer	real
		real	real	real
		absolute time	absolute time	relative time
		absolute time	relative time	absolute time
		relative time	relative time	relative time
"=", "!=", "<", ">", "<=", ">="		integer, real	integer, real	Boolean
		string[a]	string	Boolean
		relative time	relative time	Boolean
		absolute time	absolute time	Boolean
"AND"	Boolean AND[b]	Boolean	Boolean	Boolean
"OR"	Boolean OR[b]	Boolean	Boolean	Boolean
"XOR"	Boolean Exclusive OR[b]	Boolean	Boolean	Boolean

[a] Any comparisons performed on strings are done so in a case-insensitive manner.

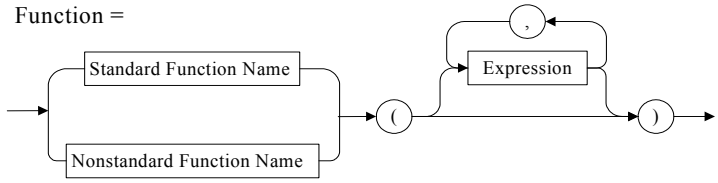
[b] Boolean expressions are evaluated in their entirety, even where the result is yielded part way through. For example, in the expression "A **AND** B", where A is "**FALSE**", the expression is "**FALSE**" without the need to evaluate B, nevertheless the full expression is evaluated. Therefore, it is important to avoid an expression such as: "X > 0 **AND** Y/X = Z" since a "division by 0" exception occurs where X = 0.

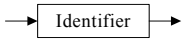
A.3.9.35 Function

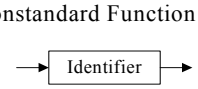
Meaning

Obtains the return value of a predefined (built-in) function operating on a list of arguments.

Syntax







Definition

Standard Function Name

A standard predefined function which is one of those given in Table C-1 (mathematical functions including functions that return the value of a mathematical or physical constant), Table C-2 (time functions) or Table C-3 (string functions).

Nonstandard Function Name

A non-standard predefined function.

Expression(s)

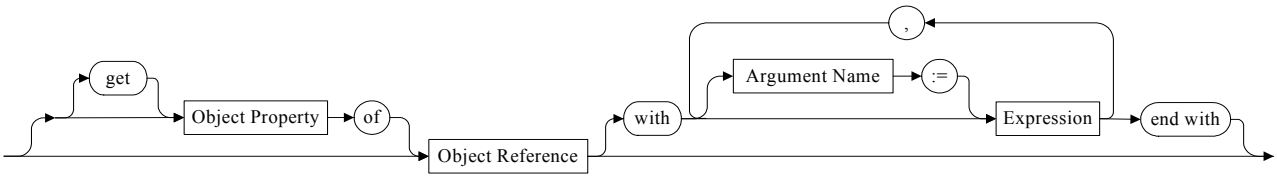
Yields the set of zero or more values to be used as arguments for the function.

A.3.9.36 Object Property Request

Meaning — Obtains the requested property of the referenced object.

Syntax

Object Property Request =



Definition

Object Property

The property that is requested, which can be either a standard property for an activity, step, reporting data, variable, procedure argument or event or a non-standard property for any object.

Object Reference

The name of the object whose property is requested.

The object may be an activity, a step, reporting data or an event. However, it may also be any system element defined within the SSM.

The standard properties that can be requested for an activity or step are listed in Table A-5, for reporting data, a variable or a procedure argument in Table A-6 and for an event in Table A-7. The availability of individual property requests, or their result types, depends on the type of the object (for example, a telecommand does not have the same result types for "execution status" as a procedure).

This language construct also provides access to any other property defined within the EMCS for these objects, or properties defined for any system element. The properties that are available depend on the particular implementation of the EMCS.

To simplify the procedure language, the following conventions apply:

- Reference to a property of an object alone implies the "**get**" request (e.g. "**if value of** Battery1 Voltage **>** 15.5 **V then**…" implies that the value of the parameter Battery1 Voltage is "got").
- Reference to a parameter alone (i.e. no request and no property) implies the "**get**" value request (e.g. "**wait until** Elevation **of** Redu Prime Antenna **<** 5 **deg** …" implies that the value of the parameter Elevation **of** Redu Prime Antenna is "got").

Table A-5: Activity and step property requests

Property request	Meaning	Argument	Result
"**get execution status**"	Gets the execution status of the activity or step	None	"**not initiated**" or "**preconditions**" or "**routing**"* or "**executing**" or "**confirmation**" or "**completed**"
"**get initiation time**"	Gets the time at which the activity or step was initiated, i.e. the time at which the preconditions check was initiated	None	absolute time
"**get start time**"	Gets the time at which the execution was started, i.e. execution started by destination application process in the case of a telecommand; starting of the main body in the case of a procedure or step	None	absolute time
"**get termination time**"	Gets the time at which the execution was terminated in the case of a procedure or step	None	absolute time
"**get confirmation status**"	Gets the confirmation status of the activity or step	None	"**not available**" or "**confirmed**" or "**not confirmed**" or "**aborted**"
"**get restart number**"	Gets the number of times the activity or step has been restarted	None	unsigned integer
"**get completion time**"	Gets the time at which the activity or step was completed, i.e. confirmation completed in the case of a procedure or step; end-to-end verification completed in the case of a telecommand	None	absolute time

NOTE 1 When requesting properties of an activity or a step, the last initiate statement is implied by default.

NOTE 2 For an activity, if a previous initiate statement is referred to, an "activity statement" is explicitly provided in the initiate statement and used in the request.

NOTE 3 When an initiate statement (activity or step) occurs within a loop, the request automatically applies to the latest iteration within the loop, i.e. previous iterations are not available.

NOTE 4 When an initiate statement (activity or step) is restarted as a result of a continuation action, a counter is available giving the number of restarts.

NOTE 5 Execution statuses indicated with a "*" are only applicable for activities of type telecommand.

Table A-6: Reporting data, variable and argument property requests

Property request	Meaning	Argument	Result
"**get validity status**"	Gets the validity status of a parameter, a variable or a procedure argument	None	"**not available**" or "**valid**" or "**invalid**"
"**get sampling time**"	Gets the sampling time of a parameter or a compound parameter or the time at which a variable or a procedure argument last received a value	None	absolute time
"**get value**"	Gets the value of a parameter, a compound parameter, a variable or a procedure argument. Only engineering values are accessible to the language.	None	any pre-defined type
"**get monitoring status**"	Gets the overall monitoring status of a parameter	None	"**not available**" or "**nominal**" or "**failed**"
"**get status consistency check status**"	Gets the status consistency check status of a parameter	None	"**not available**" or "**nominal**" or "**failed**"
"**get limit check status**"	Gets the limit check status of a parameter	None	"**not available**" or "**danger high**" or "**warning high**" or "**within limits**" or "**warning low**" or "**danger low**"
"**get delta check status**"	Gets the delta check status of a parameter	None	"**not available**" or "**nominal**" or "**failed**"
"**get expected check status**"	Gets the expected state check status of a parameter	None	"**not available**" or "**nominal**" or "**failed**"

Table A-7: Event property requests

Property request	Meaning	Argument	Result
"**get last raise time**"	Gets the time at which the event was last raised	None	absolute time

Example

Procedure name	Activate Freon Loop

PLUTO script	**procedure**
	  main
	  initiate and confirm step Heating Up
	  declare
	  Boolean Status**,**
	  real AvgeTemp **units degC**
	  end declare
	  main
	  Status **:= get validity status of** Pump **of** Freon Loop1**;**
	  AvgeTemp**:= get average value of** FreonTemp
	  with
	  StartTime **:= current time** () **–** 10 **min**,
	  EndTime**:= current time** ()
	  end with;
	  <**…...**>
	  end main
	  end step;
	  < **…...**>
	end procedure

NOTE "**average value**" is a parameter property implemented within the EMCS that corresponds to the average value over a specified time interval.	

A.4 Extended Backus-Naur form (EBNF) representation of PLUTO language constructs

A.4.1 Conventions

In this annex, the ISO extended Backus-Naur form (EBNF) is used as an alternative convention to specify the syntax of the procedure language. The complete specification of ISO EBNF is given in ISO/IEC 14977, but the salient features of the convention are summarised below.

Each syntax rule consists of a non-terminal symbol and an EBNF expression separated by an equal sign "=" and terminated with a semicolon ";", e.g.

Integer Constant = Sign, { Digit }- ;

The right-hand side of the rule is the definition of the non-terminal symbol on the left-hand side.

The EBNF expression consists of terminal symbols, non-terminal symbols and connective symbols as defined in Table A-8, separated by a comma ",".

A terminal symbol is a sequence of one or more characters forming an irreducible element of the language.

Non-printing characters such as a space or a new-line have no formal effect on the syntax as long as they appear outside of a terminal symbol.

Table A-8: EBNF symbols and meanings

Symbol	Meaning
X | Y	One of X or Y (exclusive or)
[X]	Zero or one occurrence of X
{ X }	Zero or more occurrences of X
{ X }-	One or more occurrences of X
n * X	A repetition of X exactly n times
(X Y)	Grouping construct
"**Text**"	Terminal symbol (text between double quotes representing a keyword of the procedure language). If a double quote is used inside the text, the text is enclosed instead by single quotes.

EXAMPLE 1 Confirmation Status = "**confirmed**" | "**not confirmed**" | "**aborted**";

means that Confirmation Status is defined as being either "**confirmed**" or "**not confirmed**" or "**aborted**".

EXAMPLE 2 If Statement = "**if**", Expression, "**then**", {Step Statement, "**;**"}-, ["**else**", {Step Statement, "**;**"}-], "**end if**";

means that a set of one or more step statements, each followed by a "**;**" appears after the "**then**" part of the definition. The part of the definition within the square brackets (the "**else**" part) is optional, but if present occurs once only in the "if statement".

EXAMPLE 3 Integer Constant = Sign, { Digit }- ;

means that an integer constant is defined as a sign (a plus or a minus) followed by a sequence of one or more digits.

A.4.2 PLUTO language constructs

The EBNF representations of the language constructs defined in this Standard are listed below:

```
Absolute Time Constant =
        (Year, "-", Month, "-", Day Of Month,
           "T", Hour, ":", Minute, ":", Second,
              ".", Fraction Of Second, ["Z"] )
        | (Year, "-", Day,
           "T", Hour, ":", Minute, ":", Second, ".",
           Fraction Of Second, ["Z"] );

Activity Call =
      Activity Reference,
          [("with arguments", Arguments, "end with")
          | ("with value set", Predefined Value Set Reference , "end with")],
          ["with directives", Directives, "end with"];

Activity Reference =
          Object Reference;

Activity Statement =
          Identifier;

Addition Operator =
          "+" | "-";

Argument Name =
          Identifier;

Argument Reference =
          Object Reference;

Arguments =
          ( Simple Argument  | Record Argument | Array Argument ),
          {",",  ( Simple Argument | Record Argument     | Array Argument )};

Array Argument =
          [ Argument Name ], "array",
           (( Simple Argument, {",", Simple Argument })
           | ( Record Argument, {",", Record Argument })),
           "end array";

Assignment Statement =
          Variable Reference, ":=", Expression;

Boolean Constant =
          "TRUE" | "FALSE";

Boolean Operator =
          "AND" | "OR" | "XOR";
```

```
Case Statement =
        "in case", Expression, "is", Case Tag, ":", {Step Statement, ";"}-
        {"or is", Case Tag, ":", {Step Statement, ";"}-},
        ["otherwise", ":", {Step Statement, ";"}-],
        "end case";

Case Tag =
        Comparative Expression ,
        {Boolean Operator, Comparative Expression };

Characters =
        { Digit | Letter | " " | "!" | '\"' | "#" | "$" | "%" | "&" | "'" | "("
        | ")" | "*" | "+" | "," | "-" | "." | "/" | ":" | ";" | "<" | "="| ">"
        | "?" | "@"| "[" | "\\" | "]" | "^" | "_" | "`" | "{" | "|" | "}" | "~" }-;

Comparative Expression =
        (Relational Operator, Term)
         | ("between", Term, "and", Term)
         | ("within", Constant , [? Engineering Units ? | "%"], "of", Term )
         | ("in (", Term,{",", Term}-, ")");

Confirmation Body =
        "confirmation", ( ( "if", Expression ) | Wait Statement ),
        {"then", ( ( "if", Expression) | Wait Statement ) },
        "end confirmation";

Confirmation Status =
        "confirmed" | "not confirmed" | "aborted";

Constant =
        Boolean Constant
        | Enumerated Constant
        | Integer Constant
        | Real Constant
        | String Constant
        | Absolute Time Constant
        | Relative Time Constant;

Continuation Action =
        "resume"
        | "abort"
        | "restart",
          [ Timeout  | ("max times", Expression, [Raise Event] ) ]
        | "ask user"
        | Raise Event
        | "continue"
        | "terminate";

Continuation Test =
        "in case",
        {Confirmation Status, ":", Continuation Action, ";"}-,
        "end case";

Day =
        3 * Digit;

Day Of Month =
```

```
        2 * Digit;

Days =
        {Digit}-;

Description =
        "described by", String Constant;

Digit =
        "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";

Directive Name =
        Identifier;

Directives =
        [Directive Name, ":="], Expression,
        {",", [Directive Name, ":="], Expression };

Enumerated Constant =
        '"', Characters, '"';

Enumerated Set Declaration =
        "enumerated", Set Name,
        "(", Enumerated Constant,
        {",", Enumerated Constant}, ")", [Description];

Enumerated Set Reference =
        Set Name, ["of", Object Reference];

Event Declaration =
        "event", Event Name, [Description];

Event Name =
        Identifier;

Event Reference =
        Object Reference;

Expression =
        Relational Expression, [Boolean Operator, Expression];

Factor =
        Simple Factor, ["**", Factor];

Flow Control Statement =
        If Statement
        | Case Statement
        | Repeat Statement
        | While Statement
        | For Statement;

For Statement =
        "for", Variable Reference, ":=", Expression, "to", Expression,
        ["by", Expression],
        "do", {Step Statement, ";"}-, "end for";

Fraction Of Second =
```

```
        {Digit}-;

Function =
        ( Standard Function Name | Nonstandard Function Name ),
        "(", [Expression, {",", Expression}], ")";

Hexadecimal Constant =
        Hexadecimal Symbol, {Hexadecimal Digit}-;

Hexadecimal Digit =
        Digit | "A" | "B" | "C" | "D" | "E" | "F";

Hexadecimal Symbol =
        "0x";

Hour =
        2 * Digit;

Hours =
        {Digit}-;

Identifier =
        Identifier First Word, {Identifier Subsequent Word };

Identifier First Word =
        Letter, {Letter | Digit };

Identifier Subsequent Word =
        {Letter | Digit }-;

If Statement =
        "if", Expression,
        "then", {Step Statement, ";"}-,
        ["else", {Step Statement, ";"}-],
        "end if";

Inform User Statement =
        "inform user", Expression, {",", Expression};

Initiate Activity Statement =
        "initiate", Activity Call, ["refer by", Activity Statement];

Initiate And Confirm Activity Statement =
        "initiate and confirm", Activity Call,
        ["refer by", Activity Statement],
        [Continuation Test];

Initiate And Confirm Step Statement =
        "initiate and confirm step", Step Name,
        Step Definition, "end step",
        [Continuation Test];

Initiate In Parallel Statement =
        "in parallel", ( ["until all complete"] | "until one completes" ),
        (  Initiate And Confirm Step Statement
          | Initiate And Confirm Activity Statement ), ";",
        { (  Initiate And Confirm Step Statement
```

```
        | Initiate And Confirm Activity Statement ), ";"}-,
    "end parallel";

Integer Constant =
    ( [ Sign ], {Digit}-, [? Engineering Units ?] )
    | Hexadecimal Constant;

Letter =
    "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"| "k" | "l" | "m" | "n"
    | "o" | "p" | "q" | "r" | "s"| "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A"
    | "B"| "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K"| "L" | "M"
    | "N" | "O" | "P" | "Q" | "R" | "S" | "T"| "U" | "V" | "W" | "X" | "Y"
    | "Z";

Log Statement =
    "log", Expression, {",", Expression};

Minute =
    2 * Digit;

Minutes =
    {Digit}-;

Month =
    2 * Digit;

Multiplication Operator =
    "*" | "/";

Negation Boolean Operator =
    "NOT";

Nonstandard Function Name = Identifier;

Nonstandard Object Operation Name = Identifier;

Nonstandard Object Property Name = Identifier;

Object Name =
    Identifier;

Object Operation =
    ("set", Object Property)
    |Nonstandard Object Operation Name;

Object Operation Request Statement =
    Object Operation, [["of"], Object Reference],
    [ "with", [Argument Name, ":="], Expression,
    {",", [Argument Name, ":="], Expression},
    "end with"];

Object Property =
    ( Standard Object Property Name
    | Nonstandard Object Property Name), ["of", Object Property];

Object Property Request =
    [["get"], Object Property, "of"], Object Reference,
```

```
        [ "with", [Argument Name, ":="], Expression,
        {",", [Argument Name, ":="], Expression},
        "end with"];

Object Reference =
        [ Object Type ], Object Name, {"of", Object Reference};

Object Type =
        "variable" | "predefined value set" | "activity statement"
        | "step" | "argument" | "system element"    | "reporting data"
        | "parameter" | "record" | "array" | "activity" | "event";

Preconditions Body =
        "preconditions",
        ( ( "if", Expression ) | Wait Statement ),
          {"then",
            (( "if", Expression ) | Wait Statement )},
        "end preconditions";

Predefined Type =
          "Boolean"
        | Enumerated Set Reference
        | "signed integer"
        | "unsigned integer"
        | "real"
        | "string"
        | "absolute time"
        | "relative time"
        | Property Value Set
        | Property Data Type;

Predefined Value Set Reference =
        Object Reference;

Procedure Declaration Body =
        "declare",
        Event Declaration, {",", Event Declaration},
        "end declare";

Procedure Definition =
        "procedure",
        [ Procedure Declaration Body ],
        [ Preconditions Body ],
        Procedure Main Body,
        [ Watchdog Body ],
        [ Confirmation Body ],
        "end procedure";

Procedure Main Body =
        ("main", {Procedure Statement, ";"}-, "end main")
        | {Procedure Statement, ";"}-;

Procedure Statement =
          Set Procedure Context Statement
        | Initiate In Parallel Statement
        | Initiate And Confirm Step Statement
        | Initiate And Confirm Activity Statement
```

```
        | Initiate Activity Statement
        | Inform User Statement
        | Log Statement;

Product =
        Factor, [Multiplication Operator, Product];

Property Data Type =
        "same as", Object Property, "of",
        ( Object Reference | ("current", ("system element" |
        "reporting data" | "parameter" | "activity" | "event")));

Property Value Set =
        (Object Property, "of",
        (Object Reference | ( ["current"], ("system element" |
        "reporting data" | "parameter" | "activity" | "event"), ["of",
        (( Object Property, Constant)
        | Object Reference)] )))
        |(( "system element reference" |
        "reporting data reference" | "parameter reference" |
        "activity reference" | "event reference"),
        "of", Object Property, Constant);

Raise Event =
        "raise event", Event Name;

Real Constant =
        [ Sign ], {Digit}-, [ ".", {Digit}- ],
        ["e", [ Sign ], {Digit}-],
        [? Engineering Units ?];

Record Argument =
        [Argument Name], "record", Arguments, "end record";

Relational Expression =
        Term, [Comparative Expression ];

Relational Operator =
        "=" | "!=" | "<" | ">" | "<=" | ">=";

Relative Time Constant =
        ( [Sign], (Days, "d" )
        | ( [Days, "d"],
            ( ( Hours, "h" )
            | ( [Hours, "h"],
                ( ( Minutes, "min" )
                | ( [Minutes, "min"], Seconds,
                    [".", Fraction Of Second], "s" ) ) ) ) ) )
        | ([Sign], Days, ":", Hour, ":", Minute, ":", Second,
         ":", Fraction Of Second);

Repeat Statement =
        "repeat",
        {Step Statement, ";"}-,
        "until", Expression, [ Timeout ];

Reporting Data Name =
```

```
        Identifier;

Reporting Data Reference =
        Object Reference;

Save Context =
        "save context",
        "refer to", Reporting Data Reference,
         "by", Reporting Data Name,
        {",",  "to", Reporting Data Reference,
         "by", Reporting Data Name};

Save Context Statement =
        Save Context;

Second =
        2 * Digit;

Seconds =
        {Digit}-;

Set Name =
        Identifier;

Set Procedure Context Statement =
        "in the context of", Object Reference, "do",
        {Procedure Statement, ";"}-,
         "end context";

Set Step Context Statement =
        "in the context of", Object Reference, "do",
        {Step Statement, ";"}-,
         "end context";

Sign =
        "+" | "-";

Simple Argument =
         [Argument Name, ":="],
            Expression
           | ("activity", Activity Call)
           | (("parameter" | "reporting data"), Reporting Data Reference)
           | ("system element", System Element Reference)
           | ("event", Event Reference);

Simple Factor =
          Constant
        | Argument Reference
        | Variable Reference
        | Object Property Request
        | ("ask user", "(", Expression, ["default", Expression], ")",
            ["expect", Predefined Type] )
        | ("(", Expression, ")")
        | Function
        | (Sign, Simple Factor)
        | (Negation Boolean Operator, Simple Factor);
```

```
Standard Function Name = Identifier;

Standard Object Property Name = Identifier;

Step Declaration Body =
        "declare",
        (Enumerated Set Declaration
        | Variable Declaration | Event Declaration),
        {",", (Enumerated Set Declaration
        | Variable Declaration | Event Declaration)},
        "end declare";

Step Definition =
        [ Step Declaration Body ],
        [ Preconditions Body ],
          Step Main Body,
        [ Watchdog Body ],
        [ Confirmation Body ];

Step Main Body =
        ("main", {Step Statement, ";"}-, "end main")
        | {Step Statement, ";"}-;

Step Name =
        Identifier;

Step Statement =
          Set Step Context Statement
        | Assignment Statement
        | Flow Control Statement
        | Wait Statement
        | Object Operation Request Statement
        | Save Context Statement
        | Initiate In Parallel Statement
        | Initiate And Confirm Step Statement
        | Initiate And Confirm Activity Statement
        | Initiate Activity Statement
        | Inform User Statement
        | Log Statement;

String Constant =
        '"', Characters, '"';

System Element Reference =
        Object Reference;

Term =
        Product, [Addition Operator, Term];

Timeout =
        "timeout", Expression, [Raise Event];

Variable Declaration =
        "variable", Variable Name, "of type", Predefined Type,
        ["with units", ? Engineering Units ?], [Description];

Variable Name =
```

```
        Identifier;

Variable Reference =
        Object Reference;

Wait Statement =
        (( "wait until", Expression )
          | ( "wait for", (("event",  Event Reference) | Expression))),
        [ Save Context ],
        [ Timeout ];

Watchdog Body =
        "watchdog", Initiate And Confirm Step Statement, ";",
        { ["watchdog"], Initiate And Confirm Step Statement, ";"},
        "end watchdog";

While Statement =
        "while", Expression, [ Timeout ],
        "do", {Step Statement, ";"}-,
        "end while";

Year =
        4 * Digit;
```

A.5 Index of PLUTO language constructs

Annex B (informative) Engineering units

B.1 Introduction

This annex defines the standard engineering units and their corresponding symbols. These standard engineering units and their syntax representation given in B.3 are based on Voluntocracy 2003 (see Bibliography).

B.2 Engineering units and symbols

a. An engineering unit symbol shall be an unambiguous string of case-sensitive characters with no “spaces”.

b. The standard set of “simple engineering units” shall be as specified in Table B-9.

c. Multiples and submultiples of an engineering unit shall be formed using the simple engineering unit symbol combined with a decimal prefix, as specified in Table B-10.

d. Binary multiples of the **bit** and **B** (i.e. byte) units shall use the binary prefixes specified in Table B-11.

e. “Compound engineering units” shall be formed by multiplying and dividing “simple engineering units”.

f. The standard set of “compound engineering units” shall be as specified in Table B-12.

g. The language used to express the engineering units symbols defined within this Standard shall be as specified in B.3. The major characteristics of this language are:

1. Prefix **symbols** are not used with:
   - **dB** (decibel), **AU** (astronomical unit), **pc** (parsec), **u** (atomic mass unit),
   - the time-related unit symbols **min** (minute), **h** (hour) and **d** (day).
2. The unit symbols **L** (liter), **Np** (neper), **deg** (degree), **arcmin** (1/60 degree), **arcsec** (1/3600 degree), **degC** (degree Celsius), **rad** (radian) and **sr** (steradian) can only use submultiple prefix symbols.
3. The unit symbols **t** (tonne), **B** (byte), **r** (revolution), and **Bd** (baud) cannot be used with submultiple prefix symbols.
4. Unit symbols formed from other unit symbols by multiplication are indicated by means of a dot i.e. "**.**" placed between them.
5. Unit symbols formed from other unit symbols by division are indicated by means of a solidus i.e. "**/**" or negative exponents.

6. The solidus cannot be repeated in the same compound unit unless contained within a parenthesized subexpression.
7. The grouping formed by a prefix symbol attached to a unit symbol constitutes a new inseparable symbol (forming a multiple or submultiple of the unit concerned) which can be raised to a positive or negative power and which can be combined with other unit symbols to form a compound unit symbol.
8. The grouping formed by surrounding compound unit symbols with parentheses ("(" and ")") constitutes a new inseparable symbol which can be raised to a positive or negative power and wich can be combined with other unit symbols to form compound unit symbols.
9. Compound prefix symbols cannot be used, i.e. prefix symbols formed by the juxtaposition of two or more prefix symbols.
10. A unit exponent follows the unit, separated by a circumflex, i.e. "^".
11. Exponents are positive or negative.
12. Fractional exponents are parenthesized.

h. If a mission uses engineering units additional to those defined in this annex, the corresponding engineering units symbols shall comply with the language defined in B.3.

Table B-9: Simple engineering units

Quantity	Name	Symbol	Definition	Correspondence to other units
length	metre	**m**	SI base unit	
	astronomical unit	**AU**	1 AU ≈ 1,495 978 70 × 10^{-11} m	
	parsec	**pc**	1 pc ≈ 206 265 AU	
volume	litre	**L**	1 L = 10^{-3} m^3	
mass	gram	**g**	1g = 10^{-3} kg (SI base unit)	
	unified atomic mass unit	**u**	1 u ≈ 1,660 538 73 × 10^{-27} kg	
	ton	**t**	1 t = 10^3 kg (SI base unit)	
time	second	**s**	SI base unit	
	minute	**min**	1 min = 60 s	
	hour	**h**	1 h = 60 min = 3 600 s	
	day	**d**	1 d = 24 h = 86 400 s	
electric current	ampere	**A**	SI base unit	
temperature	kelvin	**K**	SI base unit	
	degree Celsius	**degC**	1 °C = 1 K + 273,15	
amount of substance	mole	**mol**	SI base unit	
luminous intensity	candela	**cd**	SI base unit	
plane angle	radian	**rad**	m • m^{-1} = 1	
	revolution	**r**	1 r = 8 × atan(1) × 1 rad	
	degree	**deg**	1° = (π/180) rad	

Quantity	Name	Symbol	Definition	Correspondence to other units
	arcminute	**arcmin**	$1' = (1/60)^{o} = (\pi/10\ 800)$ rad	
	arcsecond	**arcsec**	$1'' = (1/3\ 600)^{o} = (\pi/648\ 000)$ rad	
solid angle	steradian	**sr**	$m^2 \bullet m^{-2} = 1$	1 **sr** = 1 **rad^2**
frequency	hertz	**Hz**	s^{-1}	
force	newton	**N**	$m \bullet kg \bullet s^{-2}$	
pressure	pascal	**Pa**	$m^{-1} \bullet kg \bullet s^{-2}$	1 **Pa** = 1 **N/m^2**
	bar	**bar**	1 bar = 10^5 Pa	
energy, work, quantity of heat	joule	**J**	$m^2 \bullet kg \bullet s^{-2}$	1 **J** = 1 **N.m**
energy	electron volt	**eV**	1 eV ≈ 1,602 176 462 × 10^{-19} J	
power, radiant flux	watt	**W**	$m^2 \bullet kg \bullet s^{-3}$	1**W** = 1 **J/s**
electric charge	coulomb	**C**	$s \bullet A$	
electric potential difference, electromotive force	volt	**V**	$m^2 \bullet kg \bullet s^{-3} \bullet A^{-1}$	1 **V** = 1 **W/A**
capacitance	farad	**F**	$m^{-2} \bullet kg^{-1} \bullet s^4 \bullet A^2$	1 **F** = 1 **C/V**
electrical resistance	ohm (Ω)	**Ohm**	$m^2 \bullet kg \bullet s^{-3} \bullet A^{-2}$	1 **Ohm** = 1 **V/A**
electrical conductance	siemens	**S**	$m^{-2} \bullet kg^{-1} \bullet s^3 \bullet A^2$	1 **S** = 1 **A/V**
magnetic flux	weber	**Wb**	$m^2 \bullet kg \bullet s^{-2} \bullet A^{-1}$	1 **Wb** = 1 **V.s**
magnetic flux density	tesla	**T**	$kg \bullet s^{-2} \bullet A^{-1}$	1 **T** = 1 **Wb/m^2**
inductance	henry	**H**	$m^2 \bullet kg \bullet s^{-2} \bullet A^{-2}$	1 **H** = 1 **Wb/A**
luminous flux	lumen	**lm**	$m^2 \bullet m^{-2} \bullet cd = cd$	1 **lm** = 1 **cd.sr**
illuminance	lux	**lx**	$m^2 \bullet m^{-4} \bullet cd = m^{-2} \bullet cd$	1 **lx** = 1 **lm/m^2**
logarithm of power ratio	decibel	**dB**	1 dB = 1/20 × ln (10) × 1 Np	
	neper	**Np**	1 Np = 1	
radionuclide activity	becquerel	**Bq**	s^{-1}	
absorbed dose, specific energy (imparted), kerma	gray	**Gy**	$m^2 \bullet s^{-2}$	1 **Gy** = 1 **J/kg**
dose equivalent	sievert	**Sv**	$m^2 \bullet s^{-2}$	1 **Sv** = 1 **J/kg**
information capacity	bit	**bit**		
	byte	**B**	1 B = 8 bit	
transmission rate	baud	**Bd**	1 Bd = 1 bit/s	

Table B-10: Acceptable multiples and submultiples of engineering units

Factor	Name	Symbol
10^{24}	yotta	**Y**
10^{21}	zetta	**Z**
10^{18}	exa	**E**
10^{15}	peta	**P**
10^{12}	tera	**T**
10^{9}	giga	**G**
10^{6}	mega	**M**
10^{3}	kilo	**k**
10^{2}	hecto	**h**
10^{1}	deca	**da**

Factor	Name	Symbol
10^{-1}	deci	**d**
10^{-2}	centi	**c**
10^{-3}	milli	**m**
10^{-6}	micro	**u**
10^{-9}	nano	**n**
10^{-12}	pico	**p**
10^{-15}	femto	**f**
10^{-18}	atto	**a**
10^{-21}	zepto	**z**
10^{-24}	yocto	**y**

Table B-11: Acceptable multiples of binary engineering units

Factor	Name	Symbol
2^{60}	exbi	**Ei**
2^{50}	pebi	**Pi**
2^{40}	tebi	**Ti**
2^{30}	gibi	**Gi**
2^{20}	mebi	**Mi**
2^{10}	kibi	**Ki**

Table B-12: Standard compound engineering units

Quantity	Name	Symbol
area	square metre	**m^2**
volume	cubic metre	**m^3**
rotational frequency	reciprocal second	**s^-1**
velocity, speed	metre per second	**m/s**
angular velocity	radian per second	**rad/s**
	degree per second	**deg/s**
acceleration	metre per square second	**m/s^2**
wavenumber	reciprocal metre	**m^-1**
density, mass density	kilogram per cubic metre	**kg/m^3**
linear mass density	kilogram per metre	**kg/m**
momentum	kilogram metre per second	**kg.m/s**
angular momentum	kilogram square metre per second	**kg.m^2/s**
moment of inertia	kilogram square metre	**kg.m^2**
dynamic viscosity	pascal second	**Pa.s**
torque, moment of force	newton metre	**N.m**

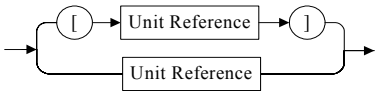
Quantity	Name	Symbol
specific acoustic impedance	pascal second per metre	**Pa.s/m**
acoustic impedance	pascal second per cubic metre	**Pa.s/m^3**
kinematic viscosity	square metre per second	**m^2/s**
volume flow rate	cubic metre per second	**m^3/s**
surface tension	newton per metre	**N/m**
linear expansion coefficient	reciprocal kelvin	**K^-1**
thermal conductivity	watt per metre kelvin	**W/(m.K)**
coefficient of heat transfer	watt per square metre kelvin	**W/(m^2.K)**
heat capacity, entropy	joule per kelvin	**J/K**
specific heat capacity, specific entropy	joule per kilogram kelvin	**J/(kg.K)**
specific energy	joule per kilogram	**J/kg**
electrical charge density	coulomb per cubic metre	**C/m^3**
electrical flux density	coulomb per square metre	**C/m^2**
electric field strength	volt per metre	**V/m**
permittivity	farad per metre	**F/m**
electric dipole moment	coulomb metre	**C.m**
current density	ampere per square metre	**A/m^2**
magnetic field strength	ampere per metre	**A/m**
electrical charge	ampere hour	**A.h**
magnetic vector potential	weber per metre	**Wb/m**
permeability	henry per metre	**H/m**
electromagnetic moment	ampere square metre	**A.m^2**
magnetization	ampere per metre	**A/m**
magnetic dipole moment	weber metre	**Wb.m**
resistivity	ohm metre	**Ohm.m**
conductivity	siemens per metre	**S/m**
reluctance	reciprocal hertz	**H^-1**
radiant intensity	watt per steradian	**W/sr**
radiance	watt per square metre steradian	**W/(m^2.sr)**
irradiance, heat flux density	watt per square metre	**W/m^2**
quantity of light	lumen second	**lm.s**
luminance	candela per square metre	**cd/m^2**
luminous exitance	lumen per square metre	**lm/m^2**
light exposure	lux second	**lx.s**
luminous efficacy	lumen per watt	**lm/W**
mechanical impedance	newton second per metre	**N.s/m**
molar mass	kilogram per mole	**kg/mol**
molar volume	cubic metre per mole	**m^3/mol**
molar energy	joule per mole	**J/mol**

Quantity	Name	Symbol
molar entropy, molar heat capacity	joule per mole kelvin	**J/(mol.K)**
concentration (of amount of substance)	mole per cubic metre	**mol/m^3**
transmission rate	bit per second	**bit/s**

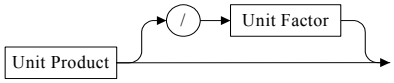
B.3 Engineering units railroad diagrams

The railroad diagrams defining the syntax of engineering units are shown below.

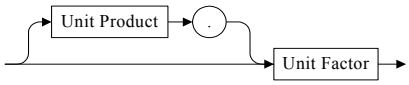
?Engineering Units? =



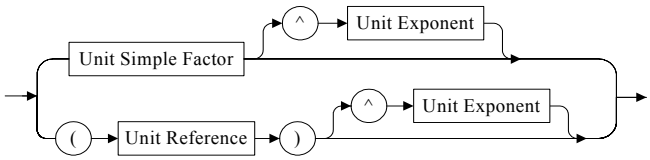
Unit Reference =



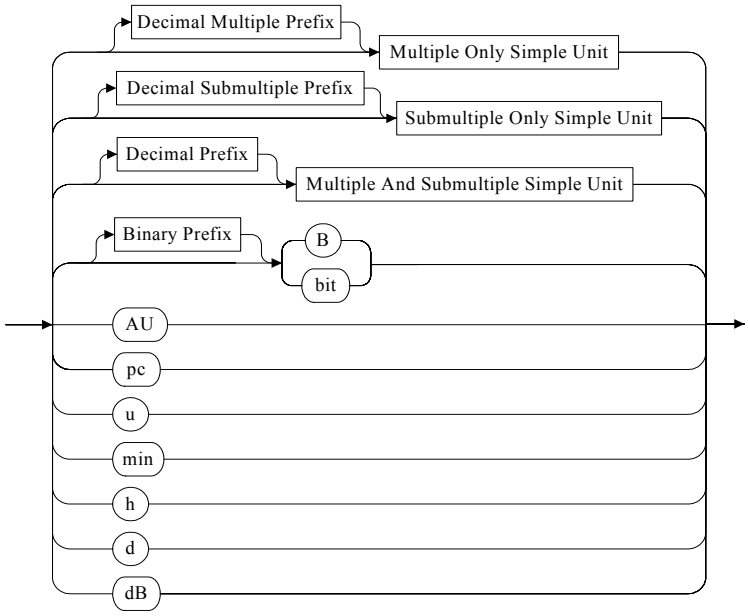
Unit Product =



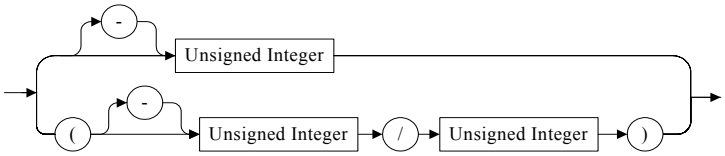
Unit Factor =



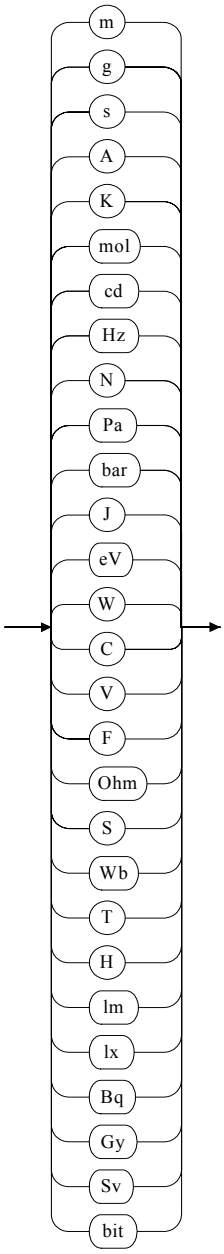
Unit Simple Factor =



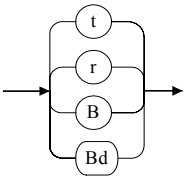
Unit Exponent =



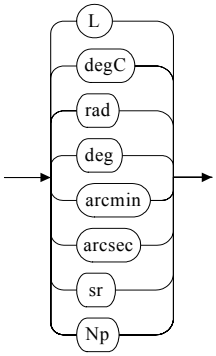
Multiple And Submultiple Simple Unit =



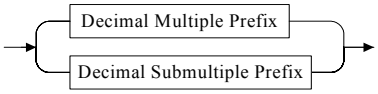
Multiple Only Simple Unit =



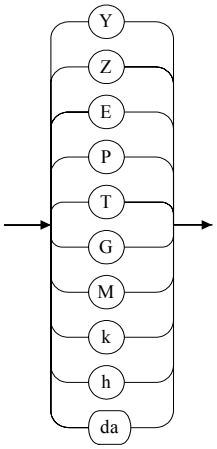
Submultiple Only Simple Unit =



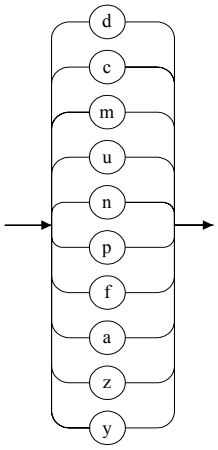
Decimal Prefix =



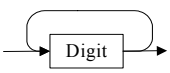
Decimal Multiple Prefix =



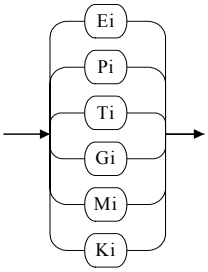
Decimal Submultiple Prefix =



Unsigned Integer =



Binary Prefix =



B.4 EBNF representation of the engineering units

The EBNF representations of the PLUTO engineering units syntax are listed below.

```
? Engineering Units ? =
        ("[",Unit Reference, "]")
        | Unit Reference;

Unit Reference =
        Unit Product,  ["/", Unit Factor];

Unit Product =
        [Unit Product, "."], Unit Factor;

Unit Factor =
        Unit Simple Factor (Unit Simple Factor , ["^", Unit Exponent])
        | ("(",Unit Reference, ")", ["^", Unit Exponent]);

Unit Simple Factor  =
        (   [Decimal Multiple Prefix], Multiple Only Simple Unit )
        | ( [Decimal Submultiple Prefix], Submultiple Only Simple Unit )
        | ( [Decimal Prefix], Multiple And Submultiple Simple Unit )
        | ( [Binary Prefix], ("B" | "bit"))
        | "AU" | "pc" | "u" |  "min" | "h" | "d" | "dB";

Unit Exponent =
        (["-"], Unsigned Integer)
        | ("(", ["-"], Unsigned Integer, "/", Unsigned Integer, ")");

Multiple And Submultiple Simple Unit =
        "m" | "g" | "s" | "A" | "K" | "mol" | "cd" |  "Hz" | "N"
        | "Pa" | "bar" | "J" | "eV" | "W" | "C" | "V" | "F" | "Ohm" | "S"
        | "Wb" | "T" | "H" | "lm" | "lx" | "Bq" | "Gy" | "Sv" | "bit";

Multiple Only Simple Unit =
        "t" | "r" | "B" | "Bd";

Submultiple Only Simple Unit =
        "L" | "degC" | "rad" | "deg" | "arcmin" | "arcsec" | "sr" | "Np";

Decimal Prefix =
        Decimal Multiple Prefix
```

```
        | Decimal Submultiple Prefix;

Decimal Multiple Prefix =
        "Y" | "Z" | "E" | "P" | "T" | "G" | "M"  | "k" | "h" | "da";

Decimal Submultiple Prefix =
        "d" | "c" | "m" | "u" | "n" | "p" | "f" | "a" | "z" | "y";

Binary Prefix =
        "Ei" | "Pi" | "Ti" | "Gi" | "Mi" | "Ki";

Unsigned Integer =
        {Digit }-;
```

(This page is intentionally left blank)

Annex C (informative) Functions

C.1 Introduction

This annex defines the standard mathematical, time and string functions.

C.2 Mathematical functions

The standard set of mathematical functions shall be as specified in Table C-1.

Table C-1: Mathematical functions

Name	Arguments	Result	Description	Examples
"**abs**"	integer or real	integer or real	Returns the absolute value of the argument.	**abs** (-9) = 9
"**acos**"	real	real	Returns the angle whose cosine is equal to the argument.	**acos** (0.5) = 1.05 **rad**
"**acosec**"	integer or real	real	Returns the angle whose cosecant is equal to the argument.	**acosec** (2) = 0.524 **rad**
"**acosec2**"	(list of 2 arguments) arg1: integer or real arg2: integer or real	real	Returns the angle, in the correct quadrant, whose cosecant is equal to the first argument divided by the second argument.	**acosec2** (-2, 1) = -0.524 **rad**
"**acotan**"	integer or real	real	Returns the angle whose cotangent is equal to the argument.	**acotan** (2) = 0.464 **rad**
"**acotan2**"	(list of 2 arguments) arg1: integer or real arg2: integer or real	real	Returns the angle, in the correct quadrant, whose cotangent is equal to the first argument divided by the second argument.	**acotan2** (-2, 1) = -0.464 **rad**
"**asec**"	integer, real	real	Returns the angle whose secant is equal to the argument.	**asec** (2) = 1.047 **rad**
"**asec2**"	(list of 2 arguments) arg1: integer or real arg2: integer or real	real	Returns the angle, in the correct quadrant, whose secant is equal to the first argument divided by the second argument.	**asec2** (-2, 1) = 2.094 **rad**
"**asin**"	real	real	Returns the angle whose sine is equal to the argument.	**asin** (0.5) = 0.52 **rad**

Name	Arguments	Result	Description	Examples
"**atan**"	integer or real	real	Returns the angle whose tangent is equal to the argument.	**atan** (1) = 0.785 **rad**
"**atan2**"	(list of 2 arguments) arg1: integer or real arg2: integer or real	real	Returns the angle, in the correct quadrant, whose tangent is equal to the first argument divided by the second argument.	**atan2** (-1, 1) = -0.785 **rad**
"**average**"	(list of 2 or more arguments) arguments: integer or real	integer or real	Returns the arithmetic average value of a list of two or more arguments.	**average** (1, 2, 3) = 2
"**ceiling**"	integer or real	integer	Returns the smallest integer value greater than or equal to the argument.	**ceiling** (5.3) = 6 **ceiling** (-5.3) = -5 **ceiling** (5) = 5
"**cos**"	integer or real	real	Returns the cosine of the argument.	**cos** (1 **rad**) = 0.54 **cos** (45 **deg**) = 0.7071
"**cosec**"	integer or real	real	Returns the cosecant of the argument.	**cosec** (1 **rad**) = 1.19
"**cosh**"	integer or real	real	Returns the hyperbolic cosine of the argument.	**cosh** (1 **rad**) = 1.54
"**cotan**"	integer or real	real	Returns the cotangent of the argument.	**cotan** (1 **rad**) = 0.64
"**floor**"	integer or real	integer	Returns the largest integer that is less than or equal to the argument.	**floor** (5.3) = 5 **floor** (-5.3) = -6
"**ln**"	integer or real	real	Returns the natural logarithm (base e) of the argument.	**ln** (1.5) = 0.405
"**log**"	integer or real	real	Returns the base 10 logarithm of the argument.	**log** (1.5) = 0.176
"**max**"	(list of 2 or more arguments) arguments: integer or real	integer or real	Returns the maximum value in a list of two or more arguments. It returns no value for just one argument.	**max** (1 **V**, 100 **mV**) = 1 **V**
"**min**"	(list of 2 or more arguments) arguments: integer or real	integer or real	Returns the minimum value in a list of two or more arguments. It returns no value for just one argument.	**min** (1, 3, 7, 4) = 1
"**quotient**"	(list of 2 arguments) arg1: integer or real arg2: integer or real	integer	Returns the result of dividing the first argument by the second argument, truncated.	**quotient** (5, 2) = 2 **quotient** (5, -2.1) = -2 **quotient** (-5, 2.1) = -2
"**remainder**"	(list of 2 arguments) arg1: integer or real arg2: integer or real	integer or real	Returns the remainder that results from dividing the first argument by the second argument.	**remainder** (5.3, 2) = 1.3
"**round**"	(1 or 2 arguments) arg1: integer or real arg2: integer or real	integer or real	Returns the first argument rounded to the number of places to the right of the decimal point given by the second argument. If the second argument is omitted, the argument is rounded to 0 places.	**round** (2.4) = 2 **round** (-2.4) = -2 **round** (2.5) = 3 **round** (2.57, 1) = 2.6
"**sec**"	integer or real	real	Returns the secant of the argument.	**sec** (1 **rad**) = 1.85

Name	Arguments	Result	Description	Examples
"**sin**"	integer or real	real	Returns the sine of the argument.	**sin** (1 **rad**) = 0.84 **sin** (45 **deg**) = 0.7071
"**sinh**"	integer or real	real	Returns the hyperbolic sine of the argument.	**sinh** (1 **rad**) = 1.18
"**sqrt**"	integer or real	integer or real	Returns the square root of the argument. It returns no value if the argument has a negative value.	**sqrt** (5) = 2.236
"**tan**"	integer or real	real	Returns the tangent of the argument.	**tan** (1 **rad**) = 1.56
"**tanh**"	integer or real	real	Returns the hyperbolic tangent of the argument.	**tanh** (1 **rad**) = 0.76
"**truncate**"	real	integer	Returns the truncated value of the argument.	**truncate** (6.6) = 6 **truncate** (-5.6) = -5
"**pi**"	none	real	Returns the value of Pythagoras' constant, π	**pi** () = 3.1415926536
"**e**"	none	real	Returns the value of Napier's constant, e (base of natural logarithm)	**e** () = 2.7182818285
"**G**"	none	real	Returns the value of the gravitational constant	**G** () = 6.6742**e**–11 **m^3.kg^-1.s^-2**

C.3 Time functions

The standard set of time functions shall be as specified in Table C-2.

Table C-2: Time functions

Name	Arguments	Result	Description	Examples
"**current time**"	none	absolute time	Returns the absolute time corresponding to the current time.	**current time** ( ) = 2004-24-21T12:00:00.0Z
"**year**"	absolute time	integer	Returns the year of the absolute time as a four-digit integer.	**year** (**current time** ( )) = 2003
"**month**"	absolute time	integer	Returns the month of the absolute time as an integer in the range 1 - 12 inclusive.	**month** (**current time** ( )) = 4
"**day of month**"	absolute time	integer	Returns the day of the month of the absolute time as an integer in the range 1 - 31 inclusive.	**day of month** (**current time** ( )) = 1
"**day of week**"	absolute time	string	Returns the day of the week of the absolute time as one of these strings: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday.	**day of week** (**current time** ( )) = "Tuesday"
"**day of year**"	absolute time	integer	Returns the day of the year of the absolute time as an integer in the range 1 - 366 inclusive.	**day of year** (**current time** ( )) = 91
"**hour**"	absolute time	integer	Returns the hour of the absolute time as	**hour** (**current time** ( )) =

			an integer in the range 0 - 23 inclusive.	11
"**minute**"	absolute time	integer	Returns the minute of the absolute time as an integer in the range 0 - 59 inclusive.	**minute** (**current time** ( )) = 11
"**second**"	absolute time	integer	Returns the second of the absolute time as an integer in the range 0 - 59 inclusive.	**second** (**current time** ( )) = 11
"**days**"	relative time	real	Returns the number of days in the relative time as a real with associated engineering units.	**days** (30 **h**) = 1.25 **d**
"**hours**"	relative time	real	Returns the number of hours in the relative time as a real with associated engineering units.	**hours** (2 **d** 5 **h** 30 **min**) = 53.5 **h**
"**minutes**"	relative time	real	Returns the number of minutes in the relative time as a real with associated engineering units.	**minutes** (2 **d** 5 **h** 30 **min**) = 3210 **min**
"**seconds**"	relative time	real	Returns the number of seconds in the relative time as a real with associated engineering units.	**seconds** (37 **min** 4.5 **s**) = 2224.5 **s**

C.4 String functions

The standard set of string functions shall be as specified in Table C-3.

Table C-3: String functions

Name	Arguments	Result	Description	Examples
"**to string**"	(1 or 2 arguments) arg1: Boolean or integer or real or absolute time or relative time arg2: engineering units	string	Returns the first argument as a string. If a second argument is given the first argument is converted to the units given by the second argument.	**to string** (5 **V**) = "5 **V** " **to string** (5 **V**, **mV**) = "5000 **mV**"
"**to Boolean**"	string	Boolean	Returns a Boolean converted from the argument.	**to Boolean** ("TRUE") = TRUE
"**to hex**"	integer	string	Returns a string, which is the hexadecimal equivalent of the argument.	**to hex** (45) = "**0x**2D"
"**to integer**"	string	integer	Returns an integer converted from the argument.	**to integer** ("32") = 32 or **to integer** ("**0x**20") = 32
"**to real**"	string	real	Returns a real converted from the argument.	**to real** ("3.2") = 3.2
"**capitalize**"	string	string	Returns a string, which is the argument with the first letter of each word capitalized.	**capitalize** ("hello world") = "Hello World"
"**get from**"	(list of 3 arguments) arg1: string arg2: integer arg3: integer	string	Returns a string, which is the string of characters, extracted from the first argument, beginning at the character	**get from** ("one two three", 5, 7) = "two"

Name	Arguments	Result	Description	Examples
			given by the second argument and ending at the character given by the third argument . Spaces between words are included in the count from left to right.	
"**insert in**"	(list of 3 arguments) arg1: string arg2: string arg3: integer	string	Returns a string, which consists of the first argument inserted into the second argument, starting at the position specified by the third argument.	**insert in** ("not ", "do enter", 4) = "do not enter"
"**is contained in**"	(list of 2 arguments) arg1: string arg2: string	Boolean	Returns "TRUE" if the second argument contains the first argument, and "FALSE" if it does not.	**is contained in** ("Your", "Your flight") = TRUE
"**length of**"	string	integer	Returns an integer whose value gives the number of characters in the argument.	**length of** ("message") = 7 **length of** ("") = 0
"**lower case**"	string	string	Returns a string, which is the same as the argument but with all alphabetic characters appearing in lower-case.	**lower case** ("123AbcDef") = "123abcdef"
"**omit from**"	(list of 3 arguments) arg1: string arg2: integer arg3: integer	string	Returns a string, which is the same as the first argument but with the range of characters, beginning at the character given by the second argument and ending at the character given by the third argument, removed.	**omit from** ("do not enter", 4, 7) = "do enter"
"**position of**"	(list of 2 arguments) arg1: string arg2: string	integer	Returns an integer value giving the starting position of the first string in the second string. If the first string occurs more than once in the second string, the first occurrence is returned. If the first string is not in the second string, the function returns "0".	**position of** ("fli", "Your flight") = 6
"**upper case**"	string	string	Returns a string, which is the same as the argument but with all alphabetic characters in upper- case.	**upper case** ("123Abc Def"= "123ABCDEF"

(This page is intentionally left blank)

Bibliography

ECSS-E-00	Space Engineering - Policy and principles
ECSS-E-10A	Space engineering - System engineering
ECSS-E-10-02	Space engineering - Verification
ECSS-E-10-03	Space engineering - Testing
ECSS-E-50	Space engineering - Communication
ECSS-E-70	Space engineering - Ground systems and operations
ECSS-E-70-31 [5]	Space engineering - Ground systems and operations – Monitoring and control data definition
ECSS-E-70-41	Space engineering - Ground systems and operations – Telemetry and telecommand packet utilization
ISO/IEC 14977	Information technology - Syntactic metalanguage – Extended BNF
Voluntocracy 2003:	Representation of numerical values and SI units in character strings for information interchange (http://swiss.csail.mit.edu/~jaffer/MIXF)

[5] To be published.

(This page is intentionally left blank)

ECSS Change Request / Document Improvement Proposal

A Change Request / Document Improvement Proposal for an ECSS Standard may be submitted to the ECSS Secretariat at any time after the standard's publication using the form presented below.

This form can be downloaded in MS Word format from the ECSS Website (www.ecss.nl, in the menus: Standards - ECSS forms).

ECSS Change Request / Document Improvement Proposal

1. Originator's name: Organization: e-mail:				2. ECSS Document number: 3. Date:	
4. Number.	5. Location of deficiency clause page (e.g. 3.1 14)		6. Changes	7. Justification	8. Disposition

Filling instructions:

1. **Originator's name -** Insert the originator's name and address
2. **ECSS document number -** Insert the complete ECSS reference number (e.g. ECSS-M-00B)
3. **Date -** Insert current date
4. **Number -** Insert originator's numbering of CR/DIP (*optional*)
5. **Location -** Insert clause, table or figure number and page number where deficiency has been identified
6. **Changes -** Identify any improvement proposed, giving as much detail as possible
7. **Justification -** Describe the purpose, reasons and benefits of the proposed change
8. **Disposition -** not to be filled in (*entered by relevant ECSS Panel*)

Once completed, please send the CR/DIP by e-mail to: ecss-secretariat@esa.int

(This page is intentionally left blank)