INTERNATIONAL STANDARD

ISO 17356-6

First edition
2006-01-15

Road vehicles — Open interface for embedded automotive applications —

Part 6: OSEK/VDX Implementation Language (OIL)

Véhicules routiers — Interface ouverte pour applications automobiles embarquées —

Partie 6: Language d'exécution OSEK/VDX (OIL)

Reference number
ISO 17356-6:2006(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 17356-6 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

ISO 17356 consists of the following parts, under the general title *Road vehicles — Open interface for embedded electronic equipment*:

— *Part 1: General structure and terms, definitions and abbreviated terms;*

— *Part 2: OSEK/VDX specifications for binding OS, COM and NM;*

— *Part 3: OSEK/VDX Operating System (OS);*

— *Part 4: OSEK/VDX Communication (COM);*

— *Part 5: OSEK/VDX Network Management (NM);*

— *Part 6: OSEK/VDX Implementation Language (OIL).*

0 Introduction

0.1 General remarks

This part of ISO 17356 refers to ISO 17356-2, ISO 17356-3 and ISO 17356-4. For a better understanding of this document, the reader should be familiar with the contents of these other specifications.

0.2 Motivation

To reach the goal of portable software, this part of ISO 17356 defines a way to describe the configuration of an application.

This part of ISO 17356 only addresses a single central processing unit (CPU) in an electronic control unit (ECU), not an ECU network.

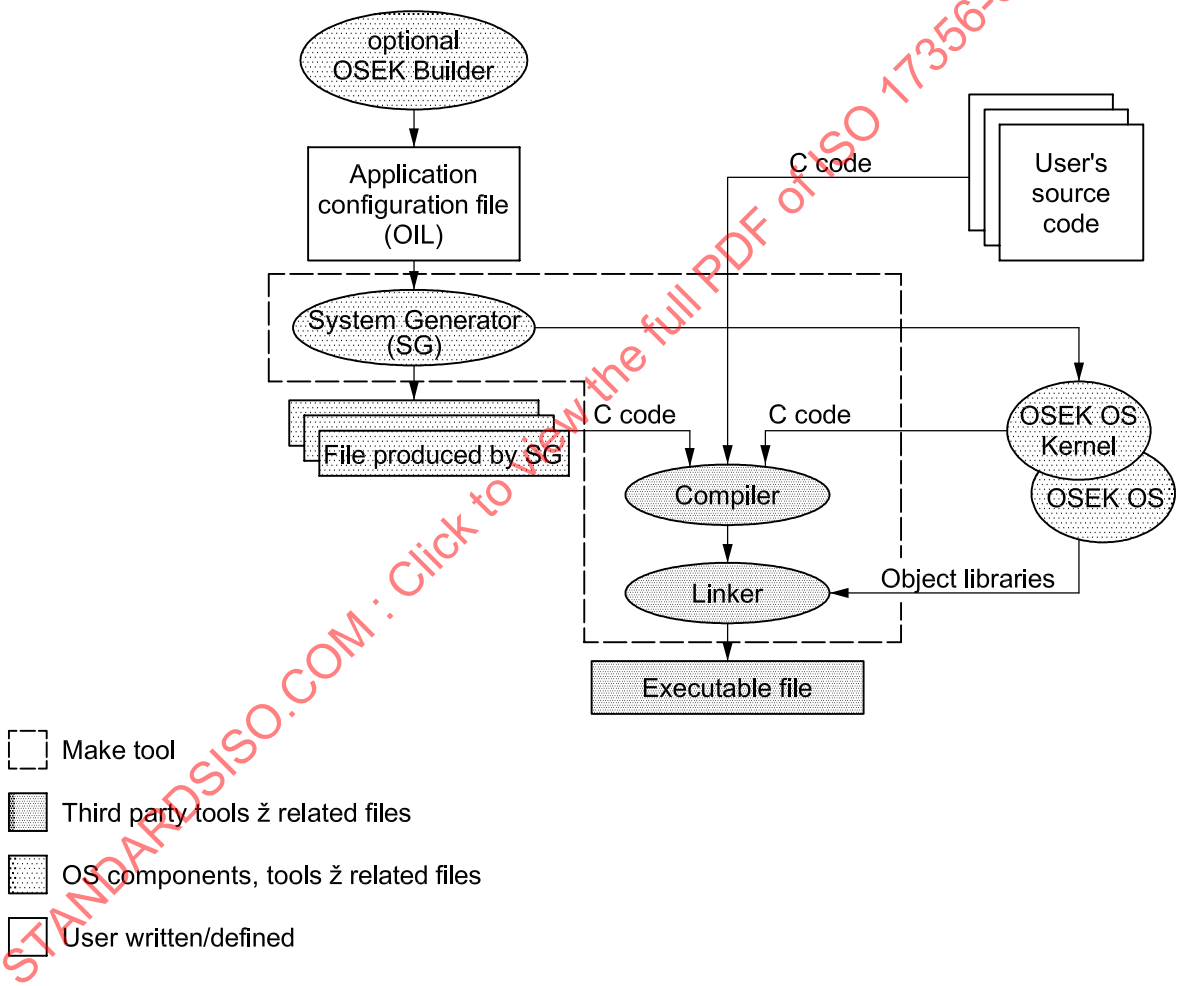

Figure 1 — Example of development process for applications

Figure 1 shows an example of a development process for applications.

The ISO 17356-6 description may be handwritten or generated by a system configuration tool. There can be several ISO 17356-6 files, e.g.:

— files which contain CPU-specific configuration items (created by the supplier); and

— files which contain configuration items for the entire network (provided by the OEM).

Sub-systems delivered in source code are compiled together with the application; others delivered as a library are integrated by the linker.

Road vehicles — Open interface for embedded automotive applications —

Part 6: OSEK/VDX Implementation Language (OIL)

1 Scope

This document describes the OSEK Implementation Language (OIL) concept for the description for ISO 17356 real-time systems, capable of multitasking and communications, which can be used for motor vehicles. It is not a product description that relates to a specific implementation.

General conventions, explanations of terms and abbreviations are compiled in a glossary, which is part of ISO 17356-1.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 9899, *Programming languages — C*

ISO 17356-1, *Road vehicles — Open interface for embedded automotive applications — Part 1: General structure and terms, definitions and abbreviated terms*

ISO 17356-2, *Road vehicles — Open interface for embedded automotive applications — Part 2: OSEK/VDX specifications for binding OS, COM and NM*

ISO 17356-3, *Road vehicles — Open interface for embedded automotive applications — Part 3: OSEK/VDX Operating System (OS)*

ISO 17356-4, *Road vehicles — Open interface for embedded automotive applications — Part 4: OSEK/VDX Communication (COM)*

ISO 17356-5, *Road vehicles — Open interface for embedded automotive applications — Part 5: OSEK/VDX Network Management (NM)*

3 Language Definition

3.1 Preamble

The goal of this part of ISO 17356 is to provide a method to configure an application inside a particular CPU. This means for each CPU there is one ISO 17356-6 description.

All system objects are described using ISO 17356-6 objects.

3.2 General concept

The ISO 17356-6 description of the application is considered to be composed of a set of ISO 17356-6 objects. A CPU is a container for these objects.

This part of ISO 17356 defines standard types for its objects. Each object is described by a set of attributes and references. This part of ISO 17356 defines explicitly all *standard attributes* for each ISO 17356-6 object.

Each implementation can define additional implementation-specific attributes and references. It is possible only to add attributes to existing ISO 17356-6 objects. Creating new ISO 17356-6 objects, or other changes to the grammar, are not allowed. All non-standard attributes (*optional attributes*) are considered to be fully implementation-specific and have no standard interpretation. Each implementation can limit the given set of values for attributes (e.g. restrict the possible value range for priorities).

3.2.1 ISO 17356-6 file structure

The ISO 17356-6 description contains two parts — one part for the definition of standard and implementation-specific features (*implementation definition*), and another for the definition of the structure of the application located on the particular CPU (*application definition*).

The ISO 17356-6 description consists of one main ISO 17356-6 file that can refer to included files (see 3.2.9).

3.2.2 Syntax

The grammar rules for an ISO 17356-6 file are presented in the document using a notation similar to the Backus-Naur Form (BNF) [1, 2], see 6.1.

All keywords, attributes, object names, and other identifiers are case-sensitive.

Comments in the BNF notation are written as C^{++}-style comments.

3.2.3 ISO 17356-6 versions

Two ISO 17356-6 sets of objects and standard attributes are defined:

— Full set of objects and standard attributes: ISO 17356-3 and full-featured ISO 17356-4, supporting the conformance classes BCC1, BCC2, ECC1, ECC2, CCCA, CCCB, CCC0, CCC1.

— Subset of objects and standard attributes: ISO 17356-3 with internal communication only, supporting the conformance classes BCC1, BCC2, ECC1, ECC2, CCCA, CCCB.

Refer to ISO 17356-3 and ISO 17356-4 for the features available with each of the abovementioned conformance classes.

3.2.4 Implementation definition

For each ISO 17356-6 object, the implementation definition defines all attributes and their properties for a particular implementation.

The implementation definition shall be present in the ISO 17356-6 description and have to contain all standard attributes, which are listed in 4.2. The value range of those attributes may be restricted. Attribute definition is described in clause 5.

Additional attributes and their properties can be defined for the objects for a particular implementation. Additional attributes are optional.

The include mechanism (see 3.2.1) can be used to define the implementation definition as a separate file. Thus, corresponding implementation definition files can be developed and delivered with particular implementations and then included with the application definition in user's ISO 17356-6 files.

An implementation of ISO 17356-6 shall support either all objects and standard attributes or a specific subset defined in 6.2.1.

3.2.5 Application definition

The application definition comprises a set of objects and the values for their attributes. Except for the ISO 17356-3, ISO 17356-4 and ISO 17356-5 objects, the application definition can contain more than one ISO 17356-6 object of a particular type.

Each object is characterized by a set of attributes and their values. No attribute may appear that is not defined in the implementation definition. Attribute values shall comply with the attribute properties specified in the implementation definition.

Attributes that take a single value shall only be specified once per object. Attributes that take a list of values shall be specified as multiple statements.

Example for a multiple statement:

```
RESOURCE = RES1;
RESOURCE = RES2;
```

3.2.6 Dependencies between attributes

This part of ISO 17356 allows the expression of dependencies between attributes. To be more open to vendor-specific and standard extensions, the ISO 17356-6 syntax includes conditional attributes (parameters). This part of ISO 17356 allows infinite nesting of those dependencies.

To express dependencies, ENUM and BOOLEAN attributes can be parameterized. If attributes in several sets of one conditional attribute have the same name, they shall have the same type.

3.2.7 Automatic attribute assignment

Attribute values may be calculated by the generator. For these attributes, the keyword WITH_AUTO shall be used in the attribute's definition in the implementation definition. In conjunction with WITH_AUTO, the attribute value AUTO is valid in the application definition and as a default value.

3.2.8 Default values

Default values are used by the generator in the case that an attribute is missing in the application definition.

Default values are mandatory for optional attributes. Because the syntax of the implementation-specific part requires the definition of default values, a special default value NO_DEFAULT is defined explicitly to suppress the default mechanism. In this case, the attribute shall be defined in the application part.

Default values are forbidden for standard attributes except if explicitly stated otherwise in the specification. If a default value is allowed for a standard attribute, it is defined in 6.2.

It is an error if a standard attribute that does not have a default value defined in the implementation definition is missing from the application definition.

ISO 17356-6 grammar uses assignment in the implementation definition to specify default values.

All possible combinations of attributes with default values are shown in Table 1. The ISO 17356-6 syntax allows six combinations for the implementation-specific part and three combinations for the application part.

Table 1 — Possible combinations of attributes with default values for ENUM

Implementation part	Application part		
	param = A;	param = AUTO;	// nothing
ENUM [A, B, C] param = B;	param ⇨A	ERROR	param ⇨B
ENUM [A, B, C] param = NO_DEFAULT;	param ⇨A	ERROR	ERROR
ENUM [A, B, C] param = AUTO;	ERROR	ERROR	ERROR
ENUM WITH_AUTO [A, B, C] param = B;	param ⇨A	generator-specific	param ⇨B
ENUM WITH_AUTO [A, B, C] param = NO_DEFAULT;	param ⇨A	generator-specific	ERROR
ENUM WITH_AUTO [A, B, C] param = AUTO;	param ⇨A	generator-specific	generator-specific

EXAMPLE

```
IMPLEMENTATION myOS {
   TASK {
   UINT32 [1..0xff] STACKSIZE = 16; // If STACKSIZE is missing,
                                    // 16 is used as a default
   };
};
```

3.2.9 Include mechanism

3.2.9.1 General

The include mechanism allows for separate definitions for some parts of this part of ISO 17356. The implementation definition can be delivered with an implementation and used (included) by the system designer.

The include statement has the same syntax as in ISO 9899:

```
#include <file>,
#include "file".
```

For each ISO 17356-6 tool there shall be a way to specify search-paths for include files.

`#include <file>` uses the search-path.

`#include "file"` uses the directory where the including file resides.

3.2.9.2 Placement of include directives

The same rules apply as for ISO 9899, e.g. the include statement has to be on a separate line and can appear anywhere in the description files.

3.2.10 Comments

The ISO 17356-6 file may contain C++-style comments (/* */ and //). C++ rules apply.

3.2.11 Descriptions

To describe ISO 17356-6 objects, attributes, and values, the ISO 17356-6 syntax offers the concept of descriptions. Descriptions are optional. They start after a colon (:), are enclosed in double quotes (" "), and shall not contain a double quote.

EXAMPLE

```
...
BOOLEAN START = FALSE:"Automatic start of alarm on system start";
...
```

Descriptions give the user additional information about ISO 17356-6 objects, attributes and values in a well-defined format. The interpretation of descriptions is implementation-specific.

4 ISO 17356-6 object definitions

4.1 Rules

The application configuration files have to conform to some rules to be successfully processed. These rules are:

— All objects are described using the ISO 17356-6 syntax.

— Each object shall have a unique name. Each object may be divided into several parts.

— All object names shall be accessible from the application.

— An attribute shall define some object properties (for example, the task priority). Attributes that take a single value may only be specified once per object. Attributes that take a list of values shall be specified as multiple statements.

— An object can have a set of references to other objects. Per object, there may be more than one reference to the same type of object (e.g. more than one reference to different events, see example in 4.2.4.8).

— Unless stated otherwise, values shall be defined for all standard attributes of all objects, except for multiple attributes, which may be empty.

— If default values are required for standard attributes, they shall be specified in this part of ISO 17356 and shall not be changed.

— The <name> non-terminal represents any ISO 9899 identifier.

— The <number> non-terminal represents any integer constant. The range of integers is determined by the target platform. Both decimal and hexadecimal integers are allowed, using the same notation as C. Decimal integers with leading zeroes are not allowed as they might be misinterpreted as octal values.

— The <string> non-terminal represents any 8-bit character sequence enclosed in double-quotes (" "), but not containing double-quotes.

— The *description* represents any 8-bit character sequence enclosed in double-quotes (" "), but not containing double-quotes.

— A *reference* defines a unidirectional link to another object (for example, the task X shall be activated when the alarm Y expires).

— Implementation-specific additional parameters are only allowed for optional attributes. For portability reasons, it is forbidden to define implementation-specific additional parameters for standard attributes.

4.2 ISO 17356-6 objects, standard attributes and references

For each object, the standard set of attributes and their values shall be defined. They shall be supported by any implementation.

4.2.1 CPU

CPU shall be used as a container for all other objects.

4.2.2 OS

OS is the object used to define ISO 17356-3 properties for an application.

In a CPU, exactly one ISO 17356-3 object shall be defined. (Attributes for Conformance Class and Scheduling are not defined, as these are not part of ISO 17356-3.)

4.2.2.1 STATUS

The STATUS attribute specifies whether a system with standard or extended status shall be used. Automatic assignment is not supported for this attribute.

This attribute is of type ENUM and has one of the following possible values:

— STANDARD,

— EXTENDED.

4.2.2.2 Hook routines

The following attribute names are defined for the hook routines supported by the OS:

— STARTUPHOOK,

— ERRORHOOK,

— SHUTDOWNHOOK,

— PRETASKHOOK,

— POSTTASKHOOK.

These attributes are of type BOOLEAN.

If a hook routine is used, the value is set to TRUE otherwise the value is set to FALSE.

The usage of the access macros to the service ID and the context-related information in the error hook is enabled by the following attributes of type BOOLEAN:

— USEGETSERVICEID,

— USEPARAMETERACCESS.

4.2.2.3 USERESSCHEDULER

The USERESSCHEDULER attribute is of type BOOLEAN and defines whether the resource RES_SCHEDULER is used within the application.

EXAMPLE

```
OS ExampleOS {
   STATUS = STANDARD;
   STARTUPHOOK = TRUE;
   ERRORHOOK = TRUE;
   SHUTDOWNHOOK = TRUE;
   PRETASKHOOK = FALSE;
   POSTTASKHOOK = FALSE;
   USEGETSERVICEID = FALSE;
   USEPARAMETERACCESS = FALSE;
   USERESSCHEDULER = TRUE;
};
```

4.2.3 APPMODE

APPMODE is the object used to define ISO 17356-3 properties for an ISO 17356-3 application mode.

No standard attributes are defined for APPMODE.

In a CPU, at least one APPMODE object shall be defined.

4.2.4 TASK

TASK objects represent tasks.

4.2.4.1 PRIORITY

The priority of a task is defined by the value of the PRIORITY attribute. This value shall be understood as a relative value, i.e. the values of PRIORITY show only the relative ordering of the tasks.

This attribute is of type UINT32.

ISO 17356-3 defines the lowest priority as zero (0); larger values of the PRIORITY attribute correspond to higher priorities.

4.2.4.2 SCHEDULE

The SCHEDULE attribute defines the preemptability of the task.

This attribute is of type ENUM and has one of the following possible values:

— NON,

— FULL.

The FULL value of this attribute corresponds to a preemptable task, the NON value to a non-preemptable task.

If the SCHEDULE attribute is set to NON, no internal resources may be assigned to this task.

4.2.4.3 ACTIVATION

The ACTIVATION attribute defines the maximum number of queued activation requests for the task. A value equal to "1" means that at any time only a single activation is permitted for this task.

This attribute is of type UINT32.

4.2.4.4 AUTOSTART

The AUTOSTART attribute determines whether or not the task is activated during the system start-up procedure for some specific application modes.

This attribute is of type BOOLEAN.

If the task is to be activated during the system start-up, the value shall be set to TRUE, otherwise the value is set to FALSE. When set to TRUE, a list of application modes is defined in the APPMODE sub-attribute of type APPMODE_TYPE. These define in which application modes the task is auto-started.

4.2.4.5 RESOURCE

The RESOURCE reference is used to define a list of resources accessed by the task.

This attribute is a multiple reference (see 5.2, 5.3) of type RESOURCE_TYPE.

4.2.4.6 EVENT

The EVENT reference is used to define a list of events the extended task may react to.

This attribute is a multiple reference (see 5.2, 5.3) of type EVENT_TYPE.

4.2.4.7 MESSAGE

The MESSAGE reference is used to define a list of messages accessed by the task.

This attribute is a multiple reference (see 5.2, 5.3) of type MESSAGE_TYPE.

4.2.4.8 Example

```
TASK TaskA {
   PRIORITY = 2;
   SCHEDULE = NON;
   ACTIVATION = 1;
   AUTOSTART = TRUE {
      APPMODE = AppMode1;
      APPMODE = AppMode2;
   };
   RESOURCE = resource1;
   RESOURCE = resource2;
   RESOURCE = resource3;
   EVENT = event1;
   EVENT = event2;
   MESSAGE = anyMesssage1;
 };
```

4.2.5 COUNTER

A COUNTER serves as a base for the ALARM mechanism.

4.2.5.1 MAXALLOWEDVALUE

The MAXALLOWEDVALUE attribute defines the maximum allowed counter value.

This attribute is of type UINT32.

4.2.5.2 TICKSPERBASE

The TICKSPERBASE attribute specifies the number of ticks required to reach a counter-specific unit. The interpretation is implementation-specific.

This attribute is of type UINT32.

4.2.5.3 MINCYCLE

The MINCYCLE attribute specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter.

This attribute is of type UINT32.

4.2.5.4 EXAMPLE

```
COUNTER Timer {
   MINCYCLE = 16;
   MAXALLOWEDVALUE = 127;
   TICKSPERBASE = 90;
};
```

4.2.6 ALARM

An ALARM may be used to asynchronously inform or activate a specific task. It is possible to start alarms automatically at system start-up depending on the application mode.

4.2.6.1 COUNTER

The COUNTER reference defines the counter assigned to this alarm. Only one counter shall be assigned to the alarm. Any alarm shall be assigned to a particular counter.

This attribute is a single reference (see 5.2).

4.2.6.2 ACTION

The ACTION attribute defines which type of notification is used when the alarm expires.

This attribute is a parameterized ENUM with the following possible values:

— ACTIVATETASK {TASK_TYPE TASK;},

— SETEVENT {TASK_TYPE TASK; EVENT_TYPE EVENT;},

— ALARMCALLBACK {STRING ALARMCALLBACKNAME;}.

For an alarm, only one action is allowed.

ACTION = ACTIVATETASK

The TASK reference parameter defines the task to be activated when the alarm expires.

This parameter is a single reference of type TASK_TYPE (see 5.2).

ACTION = SETEVENT

The TASK reference parameter defines the task for which the event is to be set. The EVENT reference parameter defines the event to be set when the alarm expires.

TASK is a single reference of type TASK_TYPE. EVENT is a single reference of type EVENT_TYPE.

ACTION = ALARMCALLBACK

The ALARMCALLBACKNAME parameter defines the name of the callback routine that is called when the alarm expires.

4.2.6.3 AUTOSTART

The AUTOSTART attribute of type BOOLEAN defines if an alarm is started automatically at system start-up depending on the application mode.

When this attribute is set to TRUE, sub-attributes are used to define the ALARMTIME, i.e. the time when the ALARM shall expire first; the CYCLETIME, i.e. the cycle time of a cyclic ALARM; and a list of application modes (APPMODE) for which the AUTOSTART shall be performed.

```
BOOLEAN [
   TRUE
   {
      UINT32 ALARMTIME;
      UINT32 CYCLETIME;
      APPMODE_TYPE APPMODE[];
   },
   FALSE
] AUTOSTART;
```

4.2.6.4 EXAMPLE

```
ALARM WakeTaskA {
   COUNTER = Timer;
   ACTION = SETEVENT {
      TASK = TaskA;
      EVENT = event1;
   };
   AUTOSTART = FALSE;
};

ALARM WakeTaskB {
   COUNTER = SysCounter;
   ACTION = ACTIVATETASK {
      TASK = TaskB;
   };
   AUTOSTART = TRUE {
      ALARMTIME = 50;
      CYCLETIME = 100;
      APPMODE = AppMode1;
      APPMODE = AppMode2;
   };
};

ALARM RunCallbackC {
   COUNTER = SysCounter;
   ACTION = ALARMCALLBACK {
      ALARMCALLBACKNAME = "CallbackC";
   };
   AUTOSTART = FALSE;
};
```

4.2.7 RESOURCE

A RESOURCE object is used to coordinate the concurrent access by tasks and ISRs to a shared resource, e.g. the scheduler, any program sequence, memory or any hardware area.

There is one attribute of type ENUM defined to specify the RESOURCEPROPERTY. This attribute can take the following values:

— STANDARD: A normal resource that is not linked to another resource and is not an internal resource.

— LINKED: A resource that is linked to another resource with the property STANDARD or LINKED. The resource to which the linking is performed shall be defined by the sub-attribute LINKEDRESOURCE of type RESOURCE_TYPE. The code generator for the operating system shall resolve chains of linked resources.

— INTERNAL: An internal resource that cannot be accessed by the application.

4.2.7.1 Example

```
RESOURCE MsgAccess
{
   RESOURCEPROPERTY = STANDARD;
};
```

4.2.8 EVENT

4.2.8.1 General

An EVENT object is represented by its mask. The name of the event is a synonym for its mask.

The same event may be set for different tasks. Events with the same name are identical; therefore, the event mask is identical. Events with the same mask are generally not identical, i.e. their names may be different.

4.2.8.2 MASK

The event mask is an integer number MASK of type UINT64. The other way to assign an event mask is to declare it as AUTO. In this case, one bit is automatically assigned to the event mask. This bit is unique with respect to the tasks that reference the event.

4.2.8.3 Examples

Following is an example of an Event:

```
EVENT event1 {
   MASK = 0x01;
};
EVENT event2 {
   MASK = AUTO;
};
```

In C Code, the user is allowed to combine normal event masks and AUTO event masks:

```
C Code:
   ...
   WaitEvent ( event1 | event2 );
   ...
```

The next example shows the same EVENT object (i.e. with the same name) used by different tasks:

```
EVENT emergency {
   MASK = AUTO;
};

TASK task1 {
   EVENT = myEvent1;
   EVENT = emergency;
};

TASK task2 {
   EVENT = emergency;
   EVENT = myEvent2;
};

TASK task7 {
   EVENT = emergency;
   EVENT = myEvent2;
};
```

In C Code, the user is allowed to use the emergency event with all three tasks:

```
C Code:
   ...
   SetEvent (task1, emergency);
   SetEvent (task2, emergency);
   SetEvent (task7, emergency);
   ...
```

Another use for the same event name for events of different tasks is in control loops:

```
C Code:
...
TaskType myList[] = {task1, task2, task7};
int myListLen = 3;
int i=0;
for (i=0;i<myListLen;i++) {
   SetEvent(myList[i],emergency);
}
...
```

4.2.9 ISR

ISR objects represent interrupt service routines (ISR).

4.2.9.1 CATEGORY

The CATEGORY attribute defines the category of the ISR.

This attribute is of type UINT32; only values of 1 and 2 are allowed.

4.2.9.2 RESOURCE

The RESOURCE reference is used to define a list of resources accessed by the ISR.

This attribute is a multiple reference (see 5.2, 5.3) of type RESOURCE_TYPE.

4.2.9.3 MESSAGE

The MESSAGE reference is used to define a list of messages accessed by the ISR.

This attribute is a multiple reference (see 5.2, 5.3) of type MESSAGE_TYPE.

4.2.9.4 EXAMPLE

```
ISR TimerInterrupt {
   CATEGORY = 2;
   RESOURCE = someResource;
   MESSAGE= anyMessage2;
 };
```

4.2.10 MESSAGE

MESSAGE objects represent messages.

To separate OEM-specific and supplier-specific attributes, the definition of messages is split into two ISO 17356-6 objects. The supplier shall configure the MESSAGE object, whereas the OEM shall configure the NETWORKMESSAGE object.

The MESSAGE object has three attributes, MESSAGEPROPERTY (see 4.2.10.1), NOTIFICATION (see 4.2.10.12) and NOTIFICATIONERROR (see 4.2.10.12).

4.2.10.1 MESSAGEPROPERTY

The MESSAGEPROPERTY attribute has the following sub-attributes:

Table 2 — Sub-attributes of MESSAGEPROPERTY

MESSAGEPROPERTY	Sub-attributes	Described in subclause
SEND_STATIC_INTERNAL	CDATATYPE	4.2.10.2
SEND_STATIC_EXTERNAL	CDATATYPE	4.2.10.2
	TRANSFERPROPERTY	4.2.10.3
	FILTER	4.2.10.5
	NETWORKORDERCALLOUT	4.2.10.6
	CPUORDERCALLOUT	4.2.10.7
	INITIALVALUE	4.2.10.8
	NETWORKMESSAGE	4.2.10.4
SEND_DYNAMIC_EXTERNAL	TRANSFERPROPERTY	4.2.10.3
	NETWORKORDERCALLOUT	4.2.10.6
	CPUORDERCALLOUT	4.2.10.7
	INITIALVALUE	4.2.10.8
	NETWORKMESSAGE	4.2.10.4
SEND_ZERO_INTERNAL	none	None
SEND_ZERO_EXTERNAL	NETWORKORDERCALLOUT	4.2.10.6
	CPUORDERCALLOUT	4.2.10.7
	NETWORKMESSAGE	4.2.10.4

Table 2 (*continued*)

MESSAGEPROPERTY	Sub-attributes	Described in subclause
RECEIVE_ZERO_INTERNAL	SENDINGMESSAGE	4.2.10.9
RECEIVE_ZERO_EXTERNAL	NETWORKORDERCALLOUT	4.2.10.6
	CPUORDERCALLOUT	4.2.10.7
	NETWORKMESSAGE	4.2.10.4
RECEIVE_UNQUEUED_INTERNAL	SENDINGMESSAGE	4.2.10.9
	FILTER	4.2.10.5
	INITIALVALUE	4.2.10.8
RECEIVE_QUEUED_INTERNAL	SENDINGMESSAGE	4.2.10.9
	FILTER	4.2.10.5
	INITIALVALUE	4.2.10.8
	QUEUESIZE	4.2.10.10
RECEIVE_UNQUEUED_EXTERNAL	CDATATYPE	4.2.10.2
	FILTER	4.2.10.5
	LINK	4.2.10.11
	INITIALVALUE	4.2.10.8
RECEIVE_QUEUED_EXTERNAL	CDATATYPE	4.2.10.2
	QUEUESIZE	4.2.10.10
	FILTER	4.2.10.5
	LINK	4.2.10.11
	INITIALVALUE	4.2.10.8
RECEIVE_DYNAMIC_EXTERNAL	LINK	4.2.10.11
	INITIALVALUE	4.2.10.8
RECEIVE_ZERO_SENDERS	CDATATYPE	4.2.10.2
	INITIALVALUE	4.2.10.8

A transmit message that is at the same time received internally and transmitted externally is declared as external (using one of the SEND_xx_EXTERNAL properties). Internal receivers of this message refer to it using the SENDINGMESSAGE attribute.

The property RECEIVE_ ZERO_SENDERS is used for messages with zero senders.

The message attributes are defined below.

4.2.10.2 CDATATYPE

The CDATATYPE attribute describes the data type of the message data using C language types (e.g. *int* or a structure name).

This attribute is of type STRING.

The purpose of this attribute is the representation of the message in a form that is meaningful to the application.

4.2.10.3 TRANSFERPROPERTY

4.2.10.3.1 General

The TRANSFERPROPERTY attribute is of type ENUM and describes the action that ISO 17356-4 takes when this message is sent by the application.

Possible values for TRANSFERPROPERTY are:

4.2.10.3.2 TRANSFERPROPERTY = TRIGGERED

The IPDU containing the message may or may not be sent immediately depending upon the IPDU's TRANSMISSIONMODE.

4.2.10.3.3 TRANSFERPROPERTY = PENDING

No action is taken.

4.2.10.3.4 TRANSFERPROPERTY = AUTO

The value defined in the TRANSFERPROPERTY attribute of the related NETWORKMESSAGE object is used.

If TRANSFERPROPERTY is defined differently in both NETWORKMESSAGE and MESSAGE objects, an error shall be generated.

If TRANSFERPROPERTY is set to AUTO in both NETWORKMESSAGE and MESSAGE objects, an error shall be generated.

4.2.10.4 NETWORKMESSAGE

The NETWORKMESSAGE reference defines the NETWORKMESSAGE that is linked to this MESSAGE.

4.2.10.5 FILTER

4.2.10.5.1 General

The FILTER attribute specifies the action of the message filter. This attribute is of type ENUM and has the following values, which are defined in the ISO 17356-4 specification.

4.2.10.5.2 FILTER = ALWAYS

This value has no sub-attributes. It is the default value for FILTER.

4.2.10.5.3 FILTER = NEVER

This value has no sub-attributes.

4.2.10.5.4 FILTER = MASKEDNEWEQUALSX

This value has the sub-attributes MASK and X.

4.2.10.5.5 FILTER = MaskedNewDiffersX

This value has the sub-attributes MASK and X.

4.2.10.5.6 FILTER = NewIsEqual

This value has no sub-attributes.

4.2.10.5.7 FILTER = NewIsDifferent

This value has no sub-attributes.

4.2.10.5.8 FILTER = MaskedNewEqualsMaskedOld

This value has the sub-attribute MASK.

4.2.10.5.9 FILTER = MaskedNewDiffersMaskedOld

This value has the sub-attribute MASK.

4.2.10.5.10 FILTER = NewIsWithin

This value has the sub-attributes MIN and MAX.

4.2.10.5.11 FILTER = NewIsOutside

This value has the sub-attributes MIN and MAX.

4.2.10.5.12 FILTER = NewIsGreater

This value has no sub-attributes.

4.2.10.5.13 FILTER = NewIsLessOrEqual

This value has no sub-attributes.

4.2.10.5.14 FILTER = NEWISLESS

This value has no sub-attributes.

4.2.10.5.15 FILTER = NewIsGreaterOrEqual

This value has no sub-attributes.

4.2.10.5.16 FILTER = ONEEVERYN

This value has the sub-attributes PERIOD and OFFSET.

4.2.10.6 NETWORKORDERCALLOUT

The NETWORKORDERCALLOUT attribute defines the name of the network-order callout routine for this MESSAGE. The default value corresponds to no callout specified.

This attribute is of type STRING.

4.2.10.7 CPUORDERCALLOUT

The CPUORDERCALLOUT attribute defines the name of the CPU-order callout routine for this MESSAGE. The default value corresponds to no callout specified.

This attribute is of type STRING.

4.2.10.8 INITIALVALUE

The INITIALVALUE attribute is of type UINT64 and specifies the initial value of a MESSAGE. If MESSAGEPROPERTY is set to RECEIVE_QUEUED_INTERNAL or RECEIVE_QUEUED_EXTERNAL, a MESSAGE has no initial value.

If a filter algorithm using a preceding value (called *old_value*) is specified for a MESSAGE, INITIALVALUE also specifies the initial value of *old_value*.

The default value for INITIALVALUE is 0.

If the MESSAGE object is related to a NETWORKMESSAGE object, the following rules shall be applied:

— If INITIALVALUE of the MESSAGE is set to AUTO, the value defined in INITIALVALUE of the related NETWORKMESSAGE object is taken as initial value of the MESSAGE.

— If INITIALVALUE of the MESSAGE is set to a value different from AUTO, this value is taken as initial value of the MESSAGE.

4.2.10.9 SENDINGMESSAGE

The SENDINGMESSAGE attribute is used by a receiver of an internal message to identify the sender of the message. Therefore, this attribute is a reference to a sent message within this ISO 17356-6 file.

4.2.10.10 QUEUESIZE

The QUEUESIZE attribute is of type UINT32 and defines the maximum number of messages that the queue for a queued message can store. The value 0 is not allowed for this attribute.

4.2.10.11 LINK

4.2.10.11.1 General

The LINK attribute is of type ENUM. It determines whether this message has its own field within the IPDU or fans out from another message's IPDU field. ISO 17356-4 allows a field in a received IPDU to correspond to one or more MESSAGE objects. When the IPDU is received, all the corresponding MESSAGE objects receive the same data.

4.2.10.11.2 LINK = TRUE

When LINK is set to TRUE, a sub-attribute called RECEIVEMESSAGE refers to another message that shall be received from the network. The links have to point to a MESSAGE with LINK set to FALSE. This implies that the field in the IPDU fans out to more than one MESSAGE object.

The RECEIVEMESSAGE sub-attribute is a reference to another MESSAGE object.

4.2.10.11.3 LINK = FALSE

When LINK is set to FALSE, the sub-attribute NETWORKMESSAGE (see 4.2.10.4) shall be defined.

The sub-attributes NETWORKORDERCALLOUT (see 4.2.10.6) and CPUORDERCALLOUT (see 4.2.10.7) may be defined.

4.2.10.12 NOTIFICATION and NOTIFICATIONERROR

4.2.10.12.1 General

The notification classes are called NOTIFICATION and NOTIFICATIONERROR. Depending on the message property, this is either a send or a receive notification. Each notification class is defined as an ENUM with the following values:

4.2.10.12.2 NONE

No notification is performed. This is the default value for NOTIFICATION and NOTIFICATIONERROR.

4.2.10.12.3 ACTIVATETASK

To perform the required notification, a task is activated. The task is named by the TASK sub-attribute, which is a reference to a TASK object.

4.2.10.12.4 SETEVENT

To perform the required notification an event is set for a task. The event and task are named by the EVENT and TASK sub-attributes.

The EVENT sub-attribute is a reference to an EVENT object. The TASK sub-attribute is a reference to a TASK object.

4.2.10.12.5 ISO 17356-4 CALLBACK

To perform the required notification, a callback routine is called. The name of the callback routine is specified in the CALLBACKROUTINENAME sub-attribute. The MESSAGE sub-attribute shall list all the messages that are sent and/or received by this callback routine.

4.2.10.12.6 FLAG

To perform the required notification, a FLAG is set. The flag is named using the FLAGNAME sub-attribute, which is of type STRING.

4.2.10.12.7 INMCALLBACK

To perform the required notification, a callback routine in an ISO 17356-5 sub-system is called. The name of the callback routine is specified with the CALLBACKROUTINENAME sub-attribute. The callback routine is called with a parameter specified by the MONITOREDIPDU sub-attribute, which is of type UINT32, but shall be within the range 0 to 65535 inclusive. MONITOREDIPDU allows the IPDU to be identified to the ISO 17356-5 sub-system.

Both of these attributes are defined as WITH_AUTO so that the system configuration tool can automatically create values consistent between ISO 17356-4 and ISO 17356-5 if it is able to.

4.2.10.13 Example

An example of the above:

```
MESSAGE myMess1 {
      MESSAGEPROPERTY = SEND_STATIC_EXTERNAL {
            CDATATYPE = "long";
            TRANSFERPROPERTY = PENDING;
            NETWORKMESSAGE = NWM_myMess1;
            FILTER = NewIsWithin {
                  MAX = 0x1234;
                  MIN = 0x12;
            };
            INITIALVALUE = 0x12;
      };
      NOTIFICATION = FLAG {
            FLAGNAME = "myMess1_finished";
      };
};


MESSAGE speed {
      MESSAGEPROPERTY = RECEIVE_UNQUEUED_EXTERNAL {
            CDATATYPE = "long";
            LINK = FALSE {
                  NETWORKMESSAGE = NWM_speed;
                  NETWORKORDERCALLOUT = "vehicle_data_active";
            };
      };
      NOTIFICATION = ACTIVATETASK {
            TASK = speed_update;
      };
};
```

4.2.11 NETWORKMESSAGE

NETWORKMESSAGE objects represent the network-specific part of messages.

These objects shall be created by the OEM.

Any external MESSAGE object shall have a reference to the NETWORKMESSAGE object to indicate how the message is externally transmitted or received, see 4.2.10.4.

4.2.11.1 IPDU

The IPDU reference defines the IPDU that carries this NETWORKMESSAGE.

4.2.11.2 MESSAGEPROPERTY

The MESSAGEPROPERTY attribute has the following sub-attributes:

Table 3 — Sub-attributes of MESSAGEPROPERTY

MESSAGEPROPERTY	Sub-attributes	Described in subclause
STATIC	SIZEINBITS	4.2.11.3
	BITORDERING	4.2.11.4
	BITPOSITION	4.2.11.5
	DATAINTERPRETATION	4.2.11.6
	INITIALVALUE	4.2.11.7
	DIRECTION	4.2.11.8
DYNAMIC	MAXIMUMSIZEINBITS	4.2.11.9
	BITORDERING	4.2.11.4
	BITPOSITION	4.2.11.5
	INITIALVALUE	4.2.11.7
	DIRECTION	4.2.11.8
ZERO	none	none

4.2.11.3 SIZEINBITS

The SIZEINBITS attribute is of type UINT32 and specifies, in bits, the size of a static-length message in an IPDU. The value 0 is not allowed for this attribute.

4.2.11.4 BITORDERING

The BITORDERING attribute is of type ENUM and specifies the bit ordering within a message. Possible values for BITORDERING are BIGENDIAN and LITTLEENDIAN.

4.2.11.5 BITPOSITION

The BITPOSITION attribute is of type UINT32.

If the BITORDERING attribute is specified as BIGENDIAN, BITPOSITION indicates the bit position of the most significant bit of the message in an IPDU.

If the BITORDERING attribute is specified as LITTLEENDIAN, BITPOSITION indicates the bit position of the least significant bit of the message in an IPDU.

4.2.11.6 DATAINTERPRETATION

The DATAINTERPRETATION attribute is of type ENUM and can be specified as UNSIGNEDINTEGER or BYTEARRAY.

This attribute allows byte swapping for unsigned integer values.

4.2.11.7 INITIALVALUE

The INITIALVALUE attribute is of type UINT64 and specifies the initial value of a MESSAGE. If MESSAGEPROPERTY is set to RECEIVE_QUEUED_INTERNAL or RECEIVE_QUEUED_EXTERNAL, a MESSAGE has no initial value.

If a filter algorithm using a preceding value (called *old_value*) is specified for a MESSAGE, INITIALVALUE also specifies the initial value of *old_value*.

The default value for INITIALVALUE is 0.

4.2.11.8 DIRECTION

4.2.11.8.1 General

The DIRECTION attribute is of type ENUM. It specifies the transfer direction of the MESSAGE.

4.2.11.8.2 DIRECTION = SENT

When DIRECTION is set to SENT, a sub-attribute called TRANSFERPROPERTY can be specified. The TRANSFERPROPERTY attribute is of type ENUM and describes the action that COM takes when this message is sent by the application. Possible actions are TRIGGERED in which the IPDU containing the message may or may not be sent immediately depending upon the IPDU's TRANSMISSIONMODE or PENDING in which no action is taken.

The TRANSFERPROPERTY attribute may not only be specified in the NETWORKMESSAGE object, but also in the related MESSAGE object.

If TRANSFERPROPERTY is defined differently in both NETWORKMESSAGE and MESSAGE objects, an error shall be generated.

If TRANSFERPROPERTY is set to AUTO in both NETWORKMESSAGE and MESSAGE objects, an error shall be generated.

4.2.11.8.3 DIRECTION = RECEIVED

When DIRECTION is set to RECEIVED, no attribute can be specified.

4.2.11.9 MAXIMUMSIZEINBITS

The MAXIMUMSIZEINBITS attribute is of type UINT32 and specifies, in bits, the maximum size that a dynamic message might reach. The value 0 is not allowed for this attribute.

4.2.11.10 Example

In continuation of the example in section 4.2.10.13, the corresponding NETWORKMESSAGE objects are shown.

```
NETWORKMESSAGE NWM_myMess1 {
      IPDU = slow_CAN_traffic;
      MESSAGEPROPERTY = STATIC {
            SIZEINBITS = 17;
            BITORDERING = BIGENDIAN;
            BITPOSITION = 5;
            DATAINTERPRETATION = UNSIGNEDINTEGER;
            INITIALVALUE = 0x12;
            DIRECTION = SENT {
                  TRANSFERPROPERTY = PENDING;
            };
      };
};
```

4.2.12 COM

4.2.12.1 General

COM is the object used to define COM sub-system properties.

In a CPU object, only one ISO 17356-4 object can be defined.

4.2.12.2 COMTIMEBASE

The COMTIMEBASE attribute defines the time base for COM. This attribute is of type FLOAT. The ISO 17356-4 time base defined by COMTIMEBASE is one second multiplied by the parameter value. Any time that is specified in ISO 17356-4 is multiplied by this time base to arrive at the intended real time.

The default value for COMTIMEBASE is 0,001, which is equal to one millisecond.

4.2.12.3 Error hook routine

The following attributes are defined for the hook routine supported by COM. These attributes are of type BOOLEAN. The hook routine is used if the value is set to TRUE. The hook routine is not used if the value is set to FALSE.

— COMERRORHOOK

The usage of the access macros to the service ID and the context-related information in the error hook routine is enabled by the following attributes:

— COMUSEGETSERVICEID,

— COMUSEPARAMETERACCESS.

The default value for these parameters is FALSE.

4.2.12.4 COMSTARTCOMEXTENSION

The COMSTARTCOMEXTENSION attribute defines whether the user-supplied function *StartCOMExtension* is called from the ISO 17356-4 function *StartCOM*.

The function is called if the value is set to TRUE. The function is not called if the value is set to FALSE, which is the default value for this attribute.

4.2.12.5 COMAPPMODE

The COMAPPMODE attribute lists all ISO 17356-4 application modes that are supported.

This attribute is of type STRING and can have multiple values.

4.2.12.6 COMSTATUS

The COMSTATUS attribute defines the level of error checking.

This attribute is of type ENUM. Extended error checking is done if the value of COMSTATUS is set to COMEXTENDED. Standard error checking is done if the value of COMSTATUS is set to COMSTANDARD, which is the default value.

4.2.12.7 USE

The USE attribute defines the names of network-specific ISO 17356-6 files to read (containing NETWORKMESSAGE and IPDU objects).

This attribute is of type STRING.

4.2.12.8 Example

An example follows:

```
COM ExampleCOM {
   COMTIMEBASE = 0.001;
   COMERRORHOOK = TRUE;
   COMUSEGETSERVICEID = FALSE;
   COMUSEPARAMETERACCESS = FALSE;
   COMSTARTCOMEXTENSION = FALSE;
   COMAPPMODE = "COMNormalMode";
   COMAPPMODE = "COMDiagnosticMode";
   COMSTATUS = COMEXTENDED;
   USE = "networkfile.oil";
};
```

4.2.13 IPDU

4.2.13.1 SIZEINBITS

The SIZEINBITS attribute specifies the length of an IPDU in bits. This attribute is of type UINT32.

The given value of SIZEINBITS is rounded up to the nearest byte boundary.

4.2.13.2 IPDUPROPERTY

The IPDUPROPERTY attribute is of type ENUM and describes the direction of the IPDU transfer. Possible values are:

— SENT,

— RECEIVED.

4.2.13.3 TRANSMISSIONMODE

The TRANSMISSIONMODE attribute specifies the transmission mode. This attribute is of type ENUM. Possible values are:

— PERIODIC,

— DIRECT,

— MIXED.

TRANSMISSIONMODE is a sub-attribute of IPDUPROPERTY = SENT.

4.2.13.4 TIMEPERIOD

The TIMEPERIOD attribute defines, depending on the chosen transmission mode, the parameter I_TMP_TPD or I_TMM_TPD. This attribute is of type UINT64. The unit of the TIMEPERIOD parameter is multiples of the ISO 17356-4 time base.

TIMEPERIOD is a sub-attribute of TRANSMISSIONMODE = PERIODIC and TRANSMISSIONMODE = MIXED.

4.2.13.5 TIMEOFFSET

The TIMEOFFSET attribute defines, depending on the chosen transmission mode, the parameter I_TMP_TOF or I_TMM_TOF. This attribute is of type UINT64. The unit of the TIMEOFFSET parameter is multiples of the ISO 17356-4 time base.

The value AUTO is the default value for this attribute and means that TIMEOFFSET assumes the same value as TIMEPERIOD.

TIMEOFFSET is a sub-attribute of TRANSMISSIONMODE = PERIODIC and TRANSMISSIONMODE = MIXED.

4.2.13.6 MINIMUMDELAYTIME

The MINIMUMDELAYTIME attribute specifies, depending on the chosen transmission mode, the parameter I_TMD_MDT or I_TMM_MDT. This attribute is of type UINT64. The unit of the MINIMUMDELAYTIME parameter is multiples of the ISO 17356-4 time base.

The default value for MINIMUMDELAYTIME is 0, which means that no minimum delay time is enforced.

MINIMUMDELAYTIME is a sub-attribute of TRANSMISSIONMODE = DIRECT and TRANSMISSIONMODE = MIXED.

4.2.13.7 TIMEOUT

The TIMEOUT attribute specifies, if IPDUPROPERTY = RECEIVED, the parameter I_DM_RX_TO, or, if IPDUPROPERTY = SENT, the parameter I_DM_TMD_TO, I_DM_TMP_TO or I_DM_TMM_TO, depending on the chosen transmission mode.

This attribute is of type UINT64. The unit of the TIMEOUT parameter is multiples of the ISO 17356-4 time base. The notification of an IPDU timeout takes place per message.

The default value for TIMEOUT is 0, which is interpreted as no timeout.

TIMEOUT is a sub-attribute of IPDUPROPERTY = RECEIVED and of IPDUPROPERTY = SENT.

4.2.13.8 FIRSTTIMEOUT

The FIRSTTIMEOUT attribute specifies, if IPDUPROPERTY = RECEIVED, the parameter I_DM_FRX_TO. This attribute is of type UINT64. The unit of the FIRSTTIMEOUT parameter is multiples of the ISO 17356-4 time base.

The value AUTO is the default value for this attribute and means that FIRSTTIMEOUT assumes the same value as TIMEOUT. The value 0 is interpreted as no timeout.

FIRSTTIMEOUT is a sub-attribute of IPDUPROPERTY = RECEIVED.

4.2.13.9 IPDUCALLOUT

The IPDUCALLOUT attribute defines the name of the IPDU callout routine. The default value corresponds to no callout specified. This attribute is of type STRING.

4.2.13.10 LAYERUSED

The LAYERUSED attribute defines the underlying layer that is used. The default value corresponds to no underlying layer specified. This attribute is of type STRING.

4.2.13.11 Example

```
IPDU mySendIPDU {
      SIZEINBITS = 64;
      IPDUPROPERTY = SENT {
            TRANSMISSIONMODE = PERIODIC {
                  TIMEPERIOD = 2;
                  TIMEOFFSET = 100;
            };
            TIMEOUT = 250;
      };
      IPDUCALLOUT = "";
      LAYERUSED = "network";
};

IPDU myReceiveIPDU {
      SIZEINBITS = 64;
      IPDUPROPERTY = RECEIVED {
            TIMEOUT = 250;
            FIRSTTIMEOUT = 100;
      };
      IPDUCALLOUT = "";
      LAYERUSED = "network";
};
```

4.2.14 ISO 17356-5

ISO 17356-5 objects represent the network management sub-system. No standard attributes are defined for the ISO 17356-5 object.

5 Definition of a particular implementation

5.1 General

This part of ISO 17356 is intended to be used for the description of applications in any implementation. The implementation definition describes a set of attributes for each object and valid values for these attributes. All standard attributes shall be defined here. For standard attributes, the implementation definition can only limit the value range, but shall in no case extend the value range or change the value type. Optional attributes shall specify a default value, AUTO (if defined WITH_AUTO), or NO_DEFAULT.

5.2 Attribute types

5.2.1 General

Any implementation-specific attribute shall be defined before it is used.

The attribute type and attribute value range (if it exists) shall be defined. The range of attribute values can be defined in two ways: either the minimum and maximum allowed attribute values are defined (the [0..12] style) or the list of possible attribute values is presented. There shall not be a mix of both.

The WITH_AUTO specifier can be combined with any attribute type except for references. If WITH_AUTO is specified, the attribute can have the value AUTO and the possibility of automatic assignment by an off-line tool.

ISO 17356-6 data types are listed below. Note that these data types are not necessarily the same as the corresponding C data types.

5.2.2 UINT32

Any unsigned integer number (possibly restricted to a range of numbers, see <impl_attr_def> 5.1).

```
UINT32 [1..255] NON_SUSPENDED_TASKS;
UINT32 [0,2,3,5] FreeInterrupts;
UINT32 aNumber;
```

This data type allows expressing any 32-bit value in the range of [0..(2^{32}-1)].

5.2.3 INT32

This is any signed integer number in the range of [-2^{31}..(2^{31}-1)].

5.2.4 UINT64

This is any unsigned integer number in the range [0..(2^{64}-1)].

5.2.5 INT64

This is any signed integer number in the range [-2^{63}..(2^{63}-1)].

5.2.6 FLOAT

This is any floating point number according to IEEE-754 standard (Range: +/– 1,176E-38 to +/– 3,402E+38).

```
FLOAT [1.0 .. 25.3] ClockFrequency; // Clock frequency in MHz
```

5.2.7 ENUM

ENUM defines a list of ISO 9899 enumerators. Any enumerator from this list can be assigned to an attribute of the according type.

```
ENUM [NON, FULL] SCHEDULE;
ENUM [mon, tue, wed, thu, fri] myWeek;
```

ENUM types can be parameterized, i.e. the particular enumerators can have parameters. The parameter specification is denoted in curly braces after the enumerator. Any kind of attribute type is allowed as parameter of an enumerator.

```
ENUM [
   ACTIVATETASK {TASK_TYPE TASK;},
   SETEVENT {TASK_TYPE TASK; EVENT_TYPE EVENT;}
] ACTION;
```

5.2.8 BOOLEAN

An attribute of this type can take the values TRUE and FALSE.

```
BOOLEAN DontDoIt;
...
DontDoIt = FALSE;
```

BOOLEAN types can be parameterized, i.e. the particular Boolean values can have parameters. Parameter specifications are denoted in curly braces after an explicit enumeration of the Boolean values. Any kind of attribute type is allowed as parameter of a Boolean value.

```
BOOLEAN [
   TRUE {TASK_TYPE TASK; EVENT_TYPE EVENT;},
   FALSE {TASK_TYPE TASK;}
] IsEvent;
```

5.2.9 STRING

Any 8-bit character sequence enclosed in double-quotes, but not containing double-quotes, can be assigned to this attribute.

5.3 Reference Types

A reference type is a data type that refers to an ISO 17356-6 object, e.g. to a TASK object, to an EVENT object, to an ALARM object, etc.

Reference types can be used to establish links between objects, e.g. within an ALARM object description a reference type attribute can refer to the TASK object that is to be activated by the alarm.

The definition of a reference type specifies which type of object is referred to, e.g. the referenced objects are of type TASK, of type EVENT, of type ALARM, etc.

The reference type is taken from the referenced object (e.g. a reference to a task shall use the TASK_TYPE keyword as reference type). A reference can refer to any object.

A single reference type refers to exactly one object.

A definition of a single reference type consists of the object type to be referred followed by the symbolic name of the reference type being defined.

5.4 Multiple values

It is possible to use one attribute name to refer to a set of values of the same type. The set may be empty. For example, the EVENT attribute of a TASK object can refer to a set of events. Multiple values are allowed for all types.

A definition of a multiple reference type consists of the object type to be referred followed by the symbolic name of the reference type being defined followed by an empty pair of brackets '[]'.

Example: `EVENT_TYPE MYEVENTS[];`

A definition of a multiple attribute is the symbolic name followed by an empty pair of brackets “[]”.

Example: `INT32 InterruptNumber[];`

5.5 Example

The implementation can define some additional attributes for an ISO 17356-6 object or restrict the value range of standard attributes.

The example below shows:

— The limitation of the ENUM value range for the ISO 17356-3 attribute STATUS;

— The definition of an implementation-specific attribute NON_SUSPENDED_TASKS of type UINT32 with a value range;

— The limitation of the UINT32 value range for the standard task attribute PRIORITY;

— The default value for StackSize is set to 16;

— The limitation of the ENUM value range for the standard alarm attribute ACTION;

— The definition of an implementation-specific attribute START of type BOOLEAN for alarms;

— The definition of an implementation-specific attribute ITEMTYPE of type STRING for messages;

— The definition of a reference to MESSAGE objects for ISRs;

— The possible usage of the defined or modified attributes in the application definition; and

— Separation of the object MyTask1 into two definitions.

```
IMPLEMENTATION SpecialOS {
   OS {

      ENUM [EXTENDED] STATUS;
      UINT32 [1..255] NON_SUSPENDED_TASKS = 16;
      ...
   };

   TASK {
      UINT32 [1 .. 256] PRIORITY;    // define range of standard
                                     // attribute PRIORITY
      INT32 StackSize= 16;           // stacksize in bytes for a task
      ...
   };


   ALARM {
      ENUM [ACTIVATETASK {TASK_TYPE TASK;}] ACTION;                    // define
possible value(s) of standard attribute ACTION
      BOOLEAN START = FALSE;  // define implementation-specific
                              // attribute START of type BOOLEAN
      ...
   };


   MESSAGE {
      STRING ITEMTYPE = "";          // define implementation-specific
                                     // attribute ITEMTYPE of type STRING
      ...
};


   ISR {
      MESSAGE_TYPE RCV_MESSAGES[] = NO_DEFAULT;
                                     // define implementation-specific
                                     // attribute RCV_MESSAGES of type
                                     // 'multiple reference to objects
                                     // of type MESSAGE'
      ...
   };


}; // End IMPLEMENTATION SpecialOS
```

```
CPU ExampleCPU {
   OS MyOs {
      ...
   };

   TASK MyTask1 {
      PRIORITY = 17;
      ...
   };
   TASK MyTask1 {
      StackSize = 64;
      ...
   };
   ALARM MyAlarm1 {
      ACTION = ACTIVATETASK {
         TASK = MyTask1;
      };
      START = TRUE;
      ...
   };
   MESSAGE MyMsg1 {
      ITEMTYPE = "SensorData";
      ...
   };
   MESSAGE MyMsg2 {
      ITEMTYPE = "Acknowledge";
      ...
   };
   ISR MyIsr1 {
      RCV_MESSAGES = MyMsg1;
      RCV_MESSAGES = MyMsg2;
      ...
   };
}; // End CPU ExampleCPU
```

This example is not a complete ISO 17356-6 file; therefore, the ellipses represent missing parts.

6 Syntax and default definition

6.1 ISO 17356-6 syntax

The ISO 17356-6 file has the following structure:

```
<file> ::=
   <OIL_version>
   <implementation_definition>
   <application_definition>

<OIL_version> ::=
   "OIL_VERSION" "=" <version> <description> ";"
```

```
<version> ::= <string>

<implementation_definition> ::=
    "IMPLEMENTATION" <name> "{" <implementation_spec_list> "}"
    <description> ";"

<implementation_spec_list> ::=
    <implementation_spec>
    | <implementation_spec_list> <implementation_spec>

<implementation_spec> ::=
    <object> "{" <implementation_list> "}" <description> ";"

<object> ::=
    "OS" | "TASK" | "COUNTER" | "ALARM" | "RESOURCE" | "EVENT" | "ISR"
    | "MESSAGE" | "COM" | "NM" | "APPMODE" | "IPDU"

<implementation_list> ::=
        /* empty list */
    | <implementation_def>
    | <implementation_list> <implementation_def>

<implementation_def> ::= <impl_attr_def> | <impl_ref_def>

<impl_attr_def> ::=
    "UINT32" <auto_specifier> <number_range> <attribute_name>
         <multiple_specifier><default_number> <description> ";"
    | "INT32" <auto_specifier> <number_range> <attribute_name>
        <multiple_specifier> <default_number> <description> ";"
    | "UINT64" <auto_specifier> <number_range> <attribute_name>
        <multiple_specifier> <default_number> <description> ";"
    | "INT64" <auto_specifier> <number_range> <attribute_name>
        <multiple_specifier> <default_number>  <description> ";"
    | "FLOAT" <auto_specifier> <float_range> <attribute_name>
        <multiple_specifier> <default_float>  <description> ";"
    | "ENUM" <auto_specifier> <enumeration> <attribute_name>
        <multiple_specifier> <default_name>  <description> ";"
    | "STRING" <auto_specifier> <attribute_name>
        <multiple_specifier> <default_string>  <description> ";"
    | "BOOLEAN" <auto_specifier> <bool_values> <attribute_name>
        <multiple_specifier> <default_bool>  <description> ";"


<impl_parameter_list> ::=
    /* empty definition */
    |"{" <impl_def_list> "}"
```

```
<impl_def_list> ::=
    /* empty definition */
    | <implementation_def>
    | <implementation_def> <impl_def_list>

<auto_specifier> ::=
    /* empty definition */
    | "WITH_AUTO"

<number_range> ::=
    /* empty definition */
    | "[" <number> ".." <number> "]"
    | "[" <number_list> "]"

<number_list> ::=
    <number> | <number_list> "," <number>

<default_number> ::=
    /* empty definition */
    | "=" <number> | "=" "NO_DEFAULT" | "=" "AUTO"

<description> ::=
    /* empty definition */
    | ":" <string>

<float_range> ::=
    /* empty definition */
    | "[" <float> ".." <float> "]"

<default_float> ::=
    /* empty definition */
    | "=" <float> | "=" "NO_DEFAULT" | "=" "AUTO"

<enumeration> ::=
    "[" <enumerator_list> "]"

<enumerator_list> ::=
    <enumerator>
    | <enumerator_list> "," <enumerator>
```

```
<enumerator> ::=
        <name> <description>
    | <name> <impl_parameter_list> <description>

<bool_values> ::=
    /* empty definition */
    | "[" "TRUE" <impl_parameter_list> <description> ","
              "FALSE" <impl_parameter_list> <description> "]"

<default_name> ::=
    /* empty definition */
    | "=" <name> | "=" "NO_DEFAULT" | "=" "AUTO"

<default_string> ::=
    /* empty definition */
    | "=" <string> | "=" "NO_DEFAULT" | "=" "AUTO"

<default_bool> ::=
    /* empty definition */
    | "=" <boolean> | "=" "NO_DEFAULT" | "=" "AUTO"

<impl_ref_def> ::=
      <object_ref_type> <reference_name> <multiple_specifier> <description> ";"

<object_ref_type> ::=
    "OS_TYPE" | "TASK_TYPE" | "COUNTER_TYPE" | "ALARM_TYPE"
       | "RESOURCE_TYPE" "EVENT_TYPE" | "ISR_TYPE"
    | "MESSAGE_TYPE" | "COM_TYPE" | "NM_TYPE" | "APPMODE_TYPE"
       | "IPDU_TYPE"

<reference_name> ::= <name> | <object>

<multiple_specifier> ::=
    /* empty definition */
    | "[" "]"

<application_definition> ::=
    "CPU" <name> "{" <object_definition_list> "}" <description> ";"
```

```
<object_definition_list> ::=
    /* empty definition */
    | <object_definition>
    | <object_definition_list> <object_definition>

<object_definition> ::=
    <object_name> <description> ";"
    | <object_name> "{" <parameter_list> "}" <description> ";"

<object_name> ::= <object> <name>

<parameter_list> ::=
    /* empty definition */
    | <parameter>
    | <parameter_list> <parameter>

<parameter> ::=
    <attribute_name> "=" <attribute_value> <description> ";"

<attribute_name> ::= <name> | <object>


<attribute_value> ::=
    <name>
    | <name> "{" <parameter_list> "}"
    | <boolean>
    | <boolean> "{" <parameter_list> "}"
    | <number>
    | <float>
    | <string>
    | "AUTO"

<name> ::= Name

<string> ::= String
<boolean> ::= "FALSE" | "TRUE"

<number> ::= <dec_number> | <hex_number>
```

```
<dec_number>  ::=
    <sign> <int_digits>

<sign> ::=
    /* empty definition */
    | "+"
    | "-"

<int_digits> ::=
    <zero_digit>
    | <pos_digit>
    | <pos_digit> <dec_digits>

<dec_digits> ::=
    | <dec_digit>
    | <dec_digit> <dec_digits>

<float> ::=
    <sign> <dec_digits> "." <dec_digits> <exponent>

<exponent> ::=
    /* empty definition */
    | "e" <sign> <dec_digits>
    | "E" <sign> <dec_digits>

<zero_digit> ::=
    "0"

<pos_digit> ::=
    "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<dec_digit> ::= <zero_digit> | <pos_digit>

<hex_number> ::= "0x" <hex_digits>

<hex_digits> ::=
    <hex_digit>
    | <hex_digit> <hex_digits>
```

```
<hex_digit> ::=
    "A" | "B" | "C" | "D" | "E" | "F"
    | "a" | "b" | "c" | "d" | "e" | "f"
    | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

6.2 Default definition of ISO 17356-6 objects and standard attributes

The definition of standard attribute types and parameters can be presented in the following form (ordering of the elements is free):

```
IMPLEMENTATION Standard {

   OS {
      ENUM [STANDARD, EXTENDED] STATUS;
      BOOLEAN STARTUPHOOK;
      BOOLEAN ERRORHOOK;
      BOOLEAN SHUTDOWNHOOK;
      BOOLEAN PRETASKHOOK;
      BOOLEAN POSTTASKHOOK;
      BOOLEAN USEGETSERVICEID;
      BOOLEAN USEPARAMETERACCESS;
      BOOLEAN USERESSCHEDULER = TRUE;
   };

   APPMODE {
   };

   TASK {
      BOOLEAN [
         TRUE {
            APPMODE_TYPE APPMODE[];
         },
         FALSE
      ] AUTOSTART;
      UINT32 PRIORITY;
      UINT32 ACTIVATION;
      ENUM [NON, FULL] SCHEDULE;
      EVENT_TYPE EVENT[];
      RESOURCE_TYPE RESOURCE[];
      MESSAGE_TYPE MESSAGE[];
   };

   ISR {
      UINT32 [1, 2] CATEGORY;
      RESOURCE_TYPE RESOURCE[];
      MESSAGE_TYPE MESSAGE[];
   };

   COUNTER {
      UINT32 MINCYCLE;
      UINT32 MAXALLOWEDVALUE;
      UINT32 TICKSPERBASE;
   };

   ALARM {
      COUNTER_TYPE COUNTER;
      ENUM [
         ACTIVATETASK {
```

```
            TASK_TYPE TASK;
         },
         SETEVENT {
                        TASK_TYPE TASK;
                        EVENT_TYPE EVENT;
                  }
                  ALARMCALLBACK {
                        STRING ALARMCALLBACKNAME;
                  }
      ] ACTION;
      BOOLEAN [
         TRUE {
            UINT32 ALARMTIME;
            UINT32 CYCLETIME;
            APPMODE_TYPE APPMODE[];
         },
         FALSE
      ] AUTOSTART;
   };

   EVENT {
      UINT64 WITH_AUTO MASK;
   };

   RESOURCE {
      ENUM [
         STANDARD,
         LINKED {
            RESOURCE_TYPE LINKEDRESOURCE;
         },
         INTERNAL
      ] RESOURCEPROPERTY;
   };

MESSAGE {
      ENUM [
            SEND_STATIC_INTERNAL {
               STRING CDATATYPE;
            },
            SEND_STATIC_EXTERNAL {
                  STRING CDATATYPE;
                  ENUM WITH_AUTO [
                        TRIGGERED,
                        PENDING
                  ] TRANSFERPROPERTY = AUTO;
                  ENUM [
                        ALWAYS,
                        NEVER,
                        MASKEDNEWEQUALSX {
                              UINT64 MASK;
                              UINT64 X;
                        },
                        MASKEDNEWDIFFERSX {
                              UINT64 MASK;
                              UINT64 X;
                           },
                           NEWISEQUAL,
                           NEWISDIFFERENT,
                           MASKEDNEWEQUALSMASKEDOLD {
                              UINT64 MASK;
                           },
```

```
                    MASKEDNEWDIFFERSMASKEDOLD {
                        UINT64 MASK;
                    },
                    NEWISWITHIN {
                        UINT64 MIN;
                        UINT64 MAX;
                    },
                    NEWISOUTSIDE {
                        UINT64 MIN;
                        UINT64 MAX;
                    },
                    NEWISGREATER,
                    NEWISLESSOREQUAL,
                    NEWISLESS,
                    NEWISGREATEROREQUAL,
                    ONEEVERYN {
                        UINT64 PERIOD;
                        UINT64 OFFSET;
                    }
                ] FILTER = ALWAYS;
                STRING NETWORKORDERCALLOUT = "";
                STRING CPUORDERCALLOUT = "";
                UINT64 WITH_AUTO INITIALVALUE = AUTO;
                NETWORKMESSAGE_TYPE NETWORKMESSAGE;
        },
        SEND_DYNAMIC_EXTERNAL {
                ENUM WITH_AUTO [
                        TRIGGERED,
                        PENDING
                ] TRANSFERPROPERTY = AUTO;
                STRING NETWORKORDERCALLOUT = "";
                STRING CPUORDERCALLOUT = "";
                UINT64 WITH_AUTO INITIALVALUE = AUTO;
                NETWORKMESSAGE_TYPE NETWORKMESSAGE;
        },
        SEND_ZERO_INTERNAL {
        },
        SEND_ZERO_EXTERNAL {
                STRING NETWORKORDERCALLOUT = "";
                STRING CPUORDERCALLOUT = "";
                NETWORKMESSAGE_TYPE NETWORKMESSAGE;
        },
        RECEIVE_ZERO_INTERNAL {
                MESSAGE_TYPE SENDINGMESSAGE;
        },
        RECEIVE_ZERO_EXTERNAL {
                STRING NETWORKORDERCALLOUT = "";
                STRING CPUORDERCALLOUT = "";
                NETWORKMESSAGE_TYPE NETWORKMESSAGE;
        },
        RECEIVE_UNQUEUED_INTERNAL {
                MESSAGE_TYPE SENDINGMESSAGE;
                ENUM [
                    ALWAYS,
                        NEVER,
                        MASKEDNEWEQUALSX {
                                UINT64 MASK;
                                UINT64 X;
                        },
                        MASKEDNEWDIFFERSX {
                                UINT64 MASK;
```

```
                        UINT64 X;
                    },
                NEWISEQUAL,
                NEWISDIFFERENT,
                MASKEDNEWEQUALSMASKEDOLD {
                        UINT64 MASK;
                },
                MASKEDNEWDIFFERSMASKEDOLD {
                        UINT64 MASK;
                },
                NEWISWITHIN {
                        UINT64 MIN;
                        UINT64 MAX;
                },
                NEWISOUTSIDE {
                        UINT64 MIN;
                        UINT64 MAX;
                },
                NEWISGREATER,
                NEWISLESSOREQUAL,
                NEWISLESS,
                NEWISGREATEROREQUAL,
                ONEEVERYN {
                        UINT64 PERIOD;
                        UINT64 OFFSET;
                }
        ] FILTER = ALWAYS;
        UINT64 INITIALVALUE = 0;
    },
RECEIVE_QUEUED_INTERNAL {
        MESSAGE_TYPE SENDINGMESSAGE;
        ENUM [
            ALWAYS,
                NEVER,
                MASKEDNEWEQUALSX {
                        UINT64 MASK;
                        UINT64 X;
                },
                MASKEDNEWDIFFERSX {
                        UINT64 MASK;
                        UINT64 X;
                    },
                NEWISEQUAL,
                NEWISDIFFERENT,
                MASKEDNEWEQUALSMASKEDOLD {
                        UINT64 MASK;
                },
                MASKEDNEWDIFFERSMASKEDOLD {
                        UINT64 MASK;
                },
                NEWISWITHIN {
                        UINT64 MIN;
                        UINT64 MAX;
                },
                NEWISOUTSIDE {
                        UINT64 MIN;
                        UINT64 MAX;
                },
                NEWISGREATER,
                NEWISLESSOREQUAL,
                NEWISLESS,
```