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.

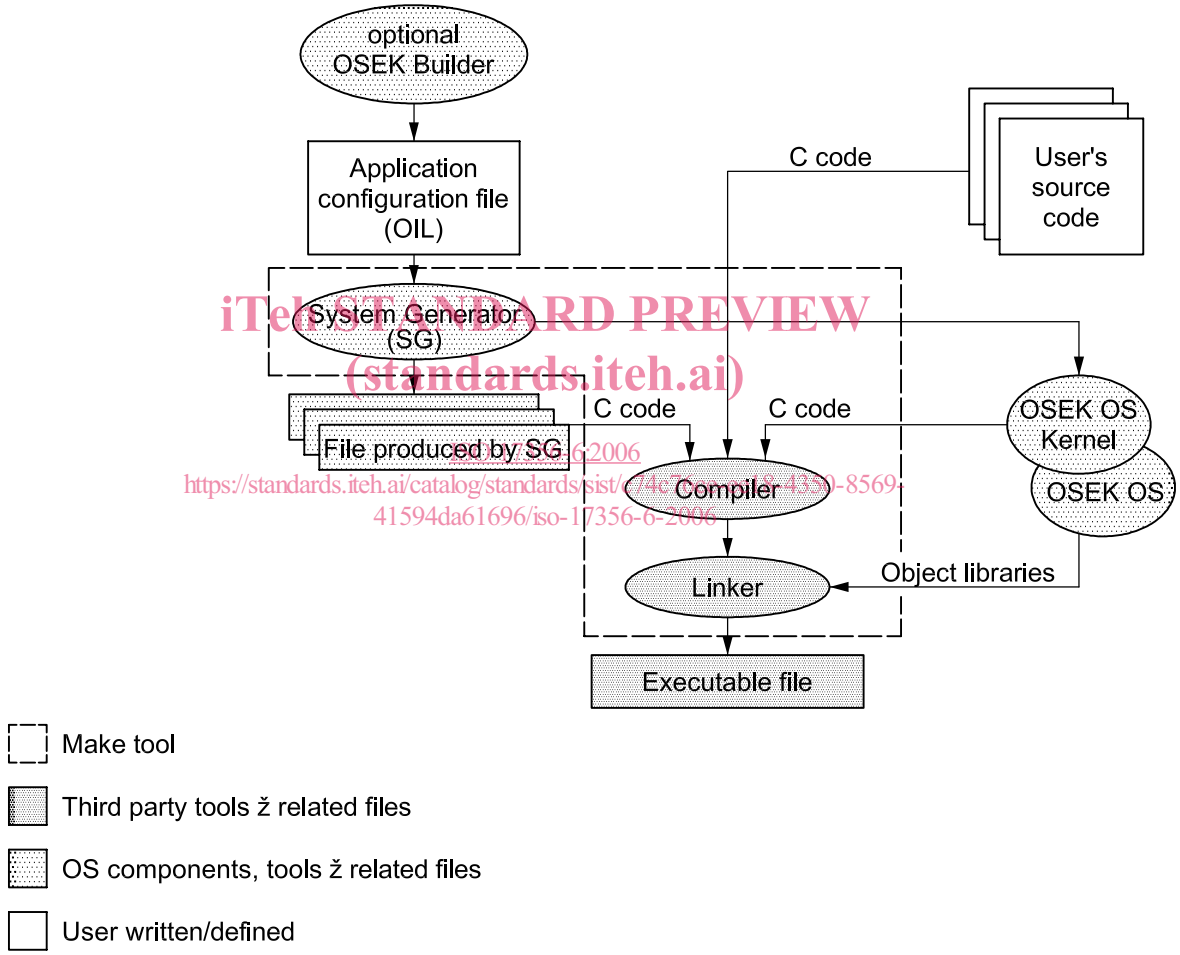

Figure 1 — Example of development process for applications

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.