FINAL
DRAFT

AMENDMENT

ISO/IEC 23001-4:2011 FDAM 1

ISO/IEC JTC **1**

Secretariat: **ANSI**

Voting begins on:
2013-02-04

Voting terminates on:
2013-04-04

Information technology — MPEG systems technologies —

Part 4: Codec configuration representation

AMENDMENT 1: RVC-CAL extensions

Technologies de l'information — Technologies des systèmes MPEG — Partie 4: Représentation de configuration code

AMENDEMENT 1: Extensions RVC-CAL

Reference number
ISO/IEC 23001-4:2011/FDAM 1:2013(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

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national 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 and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to ISO/IEC 23001-4:2011 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — MPEG systems technologies —

Part 4:
Codec configuration representation

AMENDMENT 1: RVC-CAL extensions

At the end of Clause 2, add the following paragraph:

DEFLATE Compressed Data Format Specification version 1.3. P. Deutsch, The Internet Society, May 1996.

IETF RFC 1889, *RTP A Transport Protocol for Real-Time Applications*, H. Schulzrinne, et. al., January 1996.

IETF RFC 2327, SDP: Session Description Protocol, M. Handley, April 1998.

ISO/IEC 14496-12: *Information technology– Coding of audio-visual objects – Part 12: ISO Base Media File Format* (technically identical to ISO/IEC 15444-12).

At the end of D.2, add the following paragraph:

Units. Unlike an actor, a unit does not compute anything. A unit is used to declare ‘constants’, ‘functions’, and procedures that can be referenced or imported into an actor. It cannot contain mutable variables, which would violate the design constraint that actors do not share state. Units help in factorizing the code in order not to duplicate function declarations or FU constants.

Replace the title of D.4:

D.4 Structure of actor descriptions

with:

D.4 Structure of actor/unit descriptions

After the title of D.4, add the title D.4.1:

D.4.1 Actor description

In D.4, replace the following paragraph:

Actors are the largest lexical units of specification and translation. The basic structure of an actor is this:

```
Actor → 'actor' ID '(' ActorPars ')' IOSig ':'
        {VarDecl}
        { Action | InitializationAction }
        [ ActionSchedule ]
        { PriorityBlock }
        'end'
ActorPar → Type ID [ '=' Expression ]
IOSig →[ PortDecls ] '==>' [ PortDecls ]
PortDecl → Type ID
```

with:

Actors are the largest lexical units of specification and translation. The basic structure of an actor is this:

```
Actor →('package'QualifiedName ';')?
        { Import}
        'actor' ID '(' ActorPars ')' IOSig ':'
        {VarDecl}
        { Action | InitializationAction }
        [ ActionSchedule ]
        { PriorityBlock }
        'end'
ActorPar → Type ID [ '=' Expression ]
IOSig →[ PortDecls ] '==>' [ PortDecls ]
PortDecl → {Annotation} Type ID
```


Add D.4.2:

D.4.2 Unit description

A unit can declare functions, procedures, and constants (D.6). A unit can import units. However a unit cannot import units that lead to a cyclic dependency.

```
Unit :→('package'QualifiedName ';')?
        { Import}
        'unit' ID ':'
        (FunDecl | ProcDecl|ConstantVarDecl)*
        'end'
```

Renumber D.5 – D.11 to D.6 – D.12 respectively.

Insert D.5:

D.5 Qualified names and Imports

D.5.1 Qualified names

A qualified name is represented with the following rule:

QualifiedName : ID('.' ID)*

A qualified name with a possible wildcard is allowed only in imports and is defined by:

QualifiedNameWithWildCard : QualifiedName '.*'?

D.5.2 Declaration of an entity

An entity (actor or unit) may begin with a package directive that declares the package the unit or actor resides in (a la Java). In the absence of the package declaration, the unit or actor is considered part of the “default” package, but as in Java this practice is discouraged. The qualified name of an entity is its package followed by a dot and then its identifier. In case the package is not specified, the qualified name is simply the identifier of the entity.

D.5.3 Imports

Qualified names can be imported by imports.

Import : 'import' QualifiedNameWithWildCard ';'

D.5.4 Reference to unit elements

An actor or unit may reference a variable or function declared in a unit by its **qualified name**. The qualified name of a variable or function is the name of the variable or function prefixed by the name of the unit it is declared in and a dot, e.g. MyUnit.myVar.

An actor or a unit may also **import** any or all of the variables or functions declared in a unit by using an import statement. Explicit import of one or more variables or functions is done by referencing them by their qualified name, as in:

import MyUnit.myVar;

Importing all variables or functions declared by a unit is done by using a wildcard:

import MyUnit.*;

Inside an entity, you can use either the qualified name or the simple name of the variable. If you use the simple name of a variable, this variable can be shadowed by another declaration of a variable

Add D.5.5 (D.6.5):

D.5.5 Least Upper Bound (lub)

This subclause lists the typing rules for RVC-CAL expressions.

Expression	Type of result
boolean	bool
floating-point number	float
integer with value v	type of int(v)
“xyz”	String
variable var declared with type T	T
unary expression: op e	type of unary(op, e)
binary expression: e1 op e2	type of binary(e1, op, e2)
if cond then e1 else e2 end with cond of type bool	lub(e1, e2)
list[i][j]	type of index(list, i, j)
[e1, e2, …, en : for int i1 in L1 .. H1, for int i2 in L2 .. H2, …, for int in in LN .. HN]	List(type: lub(e1, e2, …, en), size=n * (H1 – L1 + 1) * (H2 – L2 + 1) * … * (HN – LN + 1))

D.5.5.1 Least Upper Bound (lub)

The Least Upper Bound (lub) of n types is the smallest type that is compatible with the biggest of the given n types. Lub(t1, t2, …, tn) is defined as lub(…lub(lub(t1, t2), t3), …, tn).

bool, bool	bool
float, float	float
String, String	String
int(size=S1), int(size=S2)	int(size=max(S1, S2))
uint(size=S1), uint(size=S2)	uint(size=max(S1, S2))
int(size=SI), uint(size=SU) with SI > SU	int(size=SI)
int(size=SI), uint(size=SU) with SU >= SI	int(size=SU + 1)
List(type : T1, size=S1), List(type :T2, size=S2)	List(type:lub(T1, T2), size=max(S1, S2))
any other combinations	invalid

The lub is commutative: lub(t1, t2) is the same as lub(t2, t1).

D.5.5.2 Greatest Lower Bound (glb)

The Greatest Lower Bound (glb) of n types is the greatest type that is compatible with the smallest of the given n types.

bool, bool	bool
float, float	float
String, String	String
int(size=S1), int(size=S2)	int(size=min(S1, S2))
uint(size=S1), uint(size=S2)	uint(size=min(S1, S2))
int(size=SI), uint(size=SU) with SI > SU	int(size=SU + 1)
int(size=SI), uint(size=SU) with SU >= SI	int(size=SI)
any other combinations	Invalid

NOTE – glb has not been defined for List because it is not needed as typing rule.

D.5.5.3 Type of an integer

The size of an integer whose value is v is defined by the following formula:

$$\text{sizeof(v)} = \left\lceil \log 2\left(\begin{cases} v < 0 \Rightarrow -v \\ v \geq 0 \Rightarrow v+1 \end{cases}\right) \right\rceil$$

where $\lceil x \rceil$ is ceil(x), which returns the smallest integer that is not less than x.

The type of an integer whose value is v, and size is s = sizeof(v), is defined as:

If v < 0, int(size=s + 1)
If v = 0, uint(size=1)
If v > 0, uint(size=s)

D.5.5.4 Type of unary expressions

Expression	Type of result
bitnot e (or ~e) with e of type T (int or uint)	T
not e with e of type bool	bool
- e with e of type T (int or float)	T
- e with e of type uint(size=s)	int(size=s + 1)
#e with e of type List(type:T, size=S)	S
float_of_int(e) with e of type T (int or uint)	float
int_of_float(e, sz) with e of type float and sz of type int or uint	int(size=sz)
uint_of_float(e,sz) with e of type float and sz of type int or uint	uint(size=sz)

Where float_of_int, int_of_float and uint_of_float are built-ins functions for float to int/uint conversion and vice versa. The conversion to int/uint from float is the truncation conversion towards zero as used in C99 (i.e. int_of_float(5.3, 32) returns 5, and int_of_float(-5.3, 32) returns -5).

D.5.5.5 Type of binary expressions

Expression	Type of result
e1 + e2 with e1 of type String or e2 of type String	String
e1 + e2 with e1 of type List(type:T1, size=S1) e2 of type List(type:T2, size=S2)	List(type:lub(T1, T2), size=S1+S2)
e1 + e2 with e1 of type T1 (int or uint) e2 of type T2 (int or uint)	lub(T1, T2) + 1
e1 - e2 with e1 of type T1 (int or uint) e2 of type T2 (int or uint)	lub(T1, T2) + 1
e1 * e2 with e1 of type int(size=S1) or uint(size=S1) e2 of type int(size=S2) or uint(size=S2)	lub(T1, T2) with size=S1 + S2
e1 << e2 with e1 of type int(size=S1) or uint(size=S1) e2 of type int(size=S2) or uint(size=S2)	S1 + (1 << S2) – 1
e1 & e2, with e1 of type T1 (int or uint) and e2 of type T2 (int or uint)	glb(T1, T2)
e1 | e2, with e1 of type T1 (int or uint) and e2 of type T2 (int or uint)	lub(T1, T2)

Expression	Type of result
e1 ^ e2 (xor), with e1 of type T1 (int or uint) and e2 of type T2 (int or uint)	lub(T1, T2)
e1 / e2, e1 >> e2, with e1 of type T1 (int or uint) and e2 of type T2 (int or uint)	T1
e1 mod e2, with e1 of type T1 (int or uint) and e2 of type T2 (int or uint)	T2
e1 = e2, e1 != e2 with e1 of type T1 and e2 of type T2, if lub(T1, T2) exists	bool
e1 > e2, e1 >= e2, e1 < e2, e1 <= e2, with e1 of type T1 (int or uint or float) and e2 of type T2 (int or uint or float), and if lub(T1, T2) exists	bool
e1 && e2, e1 || e2, with e1 of type bool and e2 of type bool	bool
e1 + e2 with e1 of type float and e2 of type float	float
e1 - e2 with e1 of type float and e2 of type float	float
e1 * e2 with e1 of type float and e2 of type float	float
e1 / e2 with e1 of type float and e2 of type float	float

The type of binary expressions whose operator is +, -, *, /, and where one operand has type float, and the other has type int, uint, or float, is float. In other words, operands with type int or uint are automatically promoted to float.

D.5.5.6 Type of an indexing expression

The type of an indexing expression list[i1][i2]…[in] with a list of type List(type:List(type:...type:List(type:T, size=SN), size=SN_1), …, size=S1) is T if the type of i1 is not larger than the type of S1 (as obtained with sizeof(S1)), i2 is not larger than sizeof(S2), etc.

If only a subset of indexes is given, say i, then the type of the expression is the type of the i^{th} inner type.

Replace the title of D.6 (D.7):

D.6 Variables

with:

D.6 Variables, functions, and procedures

In D.6.2 (D.7.2): Explicit variable declarations, replace the following formula:

$$\text{VarDecl} \rightarrow [\text{Type}]\,\text{ID}\,[(\text{'='}\,|\,\text{':='})\,\text{Expression}]\text{';'}$$
$$|\,\text{FunDecl}\,|\,\text{ProcDecl}$$

with:

$$\text{VarDecl} \rightarrow \text{Type ID}\{\text{'['}\ \text{Expression}\text{']'}\}\ [(\text{'='}|\text{':='})\ \text{Expression}]$$

An actor may contain state variable declarations:

StateVarDecl → VarDecl ';'

A unit may contain constant variable declarations:

ConstantVarDecl → Type ID{'[' Expression']'} '=' Expression ';'

For List declaration, a more compact representation is available with an array style.

`T myVar[N1][N2]...[Nn]` – is equivalent to `List(type: List(type: ... List(type: T, size=Nn), ..., size=N2), size=N1) myVar` where the type is T.

In D.6.3 (D.7.3) Function and procedure declaration, replace the following formulas:

FormalPars →Type ID {',' Type ID }

FuncDecl →function ID '(' [FormalPars] ')' '-- >' Type [var VarDecls] ':' Expression 'end'

ProcDecl →procedure ID '(' [FormalPars] ')' [var VarDecls] 'begin' { Statement } 'end'

with:

FunDecl → 'function' ID'(' [FormalPars] ')' '-- >' Type
[['var' VarDecls] ':' Expression] 'end'

ProcDecl →'procedure' ID'(' [FormalPars] ')'
[['var' VarDecls] 'begin' { Statement }] 'end'

Renumber Annexes E and F to H and I respectively.

Insert Annexes E-G:

Annex E
(informative)

FU Classification according to their dataflow model of computation of RVC-CAL

E.1 Introduction

This Annex describes those conditions used to classify FUs, so that programmers and RVC codec implementers can make sure that an FU is classified correctly by analysis and translation tools.