ISO/IEC JTC 1/SC 29/WG 03 N1484

**ISO/IEC JTC 1/SC 29/WG 03  
MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title:** Technology under Consideration on ISO/IEC 14496-34 Syntactic Description Language

**Status:** Approved

**Date of document:** 2025-04-04

**Source:** ISO/IEC JTC 1/SC 29/WG 03

**No. of pages:** 16 (with cover page)

**Email of Convenor:** young.L @ samsung . com

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3>

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 03 MPEG SYSTEMS**

**ISO/IEC JTC 1/SC 29/WG 03 N1484**

**March 2025, online**

|  |  |
| --- | --- |
| **Title** | **Technology under Consideration on ISO/IEC 14496-34 Syntactic Description Language** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **25020** |

# Introduction

This document collects technologies being under study for consideration in the development of the standard ISO/IEC 14496-34 Syntactic Description Language

# Table of content

[1. Introduction 1](#_Toc194679793)

[2. Table of content 1](#_Toc194679794)

[3. Summary of topics 1](#_Toc194679795)

[4. Topic #1: Grammar file for SDL syntax 2](#_Toc194679796)

[5. Topic #2: On the keyword template 5](#_Toc194679797)

[6. Topic #3: On optional class member 5](#_Toc194679798)

[7. Topic #4: Array of boxes for ISOBMFF 6](#_Toc194679799)

[8. Topic #5: An unordered set of boxes 10](#_Toc194679800)

[9. Topic #6: Import directive 11](#_Toc194679801)

[10. Topic #7: SDL source file definition 12](#_Toc194679802)

[11. Topic #8: Unicode inconsistencies 12](#_Toc194679803)

# Summary of topics

|  |  |  |  |
| --- | --- | --- | --- |
| #id | Name | Description | Source |
| 1 | Grammar file | An complete grammar file describing SDL syntax. | [m62029](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/6) |
| 2 | Keyword template | Creating a new keyword inspired by the template keyword in ISOBMFF. | [m67722](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/93) |
| 3 | Optional class member | Allowing members of classes to be optional, also optional class parameters. | [m69324](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/102) |
| 4 | Array of boxes | Enabling this syntax Box boxes[]; | [m69324](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/102) |
| 5 | Unordered set of boxes | A sequence of box wherein each box is optional but appears at most once in any order. | [m69324](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/102) |
| 6 | “Imported” SDL class | Possible directive to import other SDL files into a current SDL file. | [m62014](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/4) |
| 7 | SDL file definition | A standardised SDL file format to write SDL classes into a plain text file. | [m62015](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/5) |
| 8 | Unicode inconsistencies | Inconsistencies regarding Unicode handling. | [m71608](https://git.mpeg.expert/MPEG/Systems/sdl/contributions/-/issues/108) |

# Topic #1: Grammar file for SDL syntax

## General

This clause provides the SDL grammar implementing the rules of the present specification. The grammar is based on the parsing expression grammar (PEG) concept and following the syntax and set of rules defined by the Pegen software project [1].

Currently, the grammar file is hosted an developed at <http://mpegx.int-evry.fr/software/MPEG/Systems/sdl/sdl-grammar>.

The usage of PEG seems adequate for the validation of the SDL syntax, however there are still some questions whether the PEG grammar defined by the Pegen software project is appropriate for the developing the conformance of the SDL specification. Further study is encourage on other possible alternative for such grammar file.

## Grammar file

[Editor’s note] The following is in work-in-progress and may not reflect all the rules described in the latest SDL specification text.

1. SDL grammar file

|  |
| --- |
| start: file\_input  file\_input: (NEWLINE+ | line)\* ENDMARKER  line: class\_def | comment\_cpp  #NOTE: 14496-1 forbids going back to a new line berfore {  # Rule C.1 and C.2  class\_def: aligned? abstract? 'class' NAME+ parameter\_list? parent\_class? NEWLINE? '{' body? '}'  aligned: 'aligned' '(' NUMBER+ ')'  abstract: 'abstract'  parameter\_list: '(' ','.parameter+ ')'  # NOTE: we allow arrays to be passed as parameter class, however 14496-1 is not clear on this  parameter: optional? type NAME array\_length?  # NOTE: Not in 14496-1, not sure where this come from  optional: 'optional'  # NOTE: would allow "unsigned bit", to be improved  type: signed? data\_type  signed: 'unsigned'  data\_type:  | 'bit'  | 'int'  | 'double'  array\_length: '[' NUMBER\* ']'  #NOTE: 14496-1 does not allow paramters after parent class name  parent\_class: 'extends' NAME '(' ','.value+ ')'  body: stmt\*  stmt:  | elementary\_data\_type  | non\_parsable\_variable  | assignment\_stmt  | object\_instantiation  | increment\_stmt  | if\_stmt  | switch\_stmt  | for\_stmt  | do\_stmt  | while\_stmt  | comment\_cpp  #TODO: See how to do any character up to newline  comment\_cpp: '//' (NAME | 'floor' | 'class' | 'if' | 'else' | 'for' | 'extends' | NUMBER | '==' | '=' | '{' | ';' | ',' | '-' | '/' | ':' | '?' )\*  # Rule E.1 and A.1  elementary\_data\_type: template? aligned? const? type length NAME array\_length? assigned\_value? ';'  non\_parsable\_variable: template? const? type NAME array\_length? assigned\_value? ';'  #NOTE: Not in 14496-1 but used in 14496-12  template: 'template'  const: 'const'  length: '(' (NUMBER | NAME) ')'  #NOTE: array initialisation with {val1, val2, ...} not in 14496-1  assigned\_value: '=' (value | array\_initialisation)  object\_instantiation: NAME NAME ( '(' ','.value+ ')' )\* array\_length? ';'  variable\_assignment: NAME assigned\_value  assignment\_stmt: variable\_assignment ';'  #TODO: This rule should not allow whitepaces between name and '+'s  variable\_incr: NAME '+' '+'  increment\_stmt: variable\_incr ';'  #NOTE: STRING literal e.g. 'uuid' is not allowed in 14496-1  #NOTE: NUMBER catches decimal, octal, hexadecimal, binary, foating point (scientific noation) and even imaginary number. Too broad for SDL.  value: function | expr | '-'? NUMBER | NAME | STRING  expr: (value operator value) | ( '(' value operator value ')' )  #NOTE: & and && not in 14496-1 but used in 14496-12  operator: operator\_test | operator\_logical | operator\_bin | operator\_math  operator\_math: '+' | '-' | '/' | '\*'  operator\_test: '==' | '<=' | '<' | '>=' | '>' | '!='  operator\_bin: '&' | '|'  operator\_logical: '&' '&' | '|' '|'  function: function\_name '(' value ')'  #NOTE: Only lengthof in 14496-1, floor is used in 14496-12 without definition  function\_name: 'floor' | 'lengthof'  array\_initialisation: '{' ','.value+ '}'  # Rule FC.1  if\_stmt: 'if' '(' condition ')' '{' body '}' else\_if\_stmt? else\_stmt?  else\_if\_stmt: 'else' 'if' '(' condition ')' '{' body '}'  else\_stmt: 'else' '{' body '}'  condition: value  # Rule FC.2  switch\_stmt: 'switch' '(' condition ')' '{' (switch\_case switch\_break?)\* switch\_default? switch\_break?'}'  switch\_break: 'break' ';'  switch\_case: 'case' (NUMBER | NAME | STRING) ':' body?  switch\_default: 'default' ':' body?  # Rule FC.3  for\_stmt: 'for' '(' expression1 ';' expression2 ';' expression3 ')' '{' body '}'  for\_variable\_declaration\_assignment: type NAME array\_length? assigned\_value  expression1:  | variable\_assignment  | for\_variable\_declaration\_assignment  expression2: value  expression3: variable\_incr  # Rule FC.4  do\_stmt: 'do' '{' body '}' 'while' '(' condition ')' ';'  # Rule FC.5  while\_stmt: 'while' '(' condition ')' '{' body '}' |

## References

1. Pegen documentation, <https://we-like-parsers.github.io/pegen/>

# Topic #2: On the keyword template

The template keywork in defined in ISO/IEC 14496-12 and does not belong to the original nor currently developped SDL.

The goal of the keyword is to allow other possible values than the one defined for the field by the assignment operator.

EXAMPLE ⎯

template int(32) rate = 0x00010000; // typically 1.0

In this example, the field rate shall be “0x00010000” for a file complying to this specification. But the keyword template allows a derivate spec to define another value.

From the point of a view of a file parser that only knows about the current specification, it shall throw an error is rate as a different value than “0x00010000”.

The ongoing 8ed of ISOBMFF is attempting to clarify the definition and the reader/writer behaviour.

It should be studied whether template should remain an ISOBMFF extension of the SDL or become a feature of the SDL.

# Topic #3: On optional class member

## General

Another typical case is when a box declares several fields followed some optional boxes. The meta box is also an example of that:

aligned(8) class MetaBox (handler\_type)  
 extends FullBox('meta', version = 0, 0)   
{  
 HandlerBox(handler\_type) theHandler; // optional  
 PrimaryItemBox primary\_resource[0..1]; // optional  
 DataInformationBox file\_locations; // optional  
 ItemLocationBox item\_locations; // optional  
 ItemProtectionBox protections; // optional  
 ItemInfoBox item\_infos; // optional  
 IPMPControlBox IPMP\_control; // optional  
 ItemReferenceBox item\_refs; // optional  
 ItemPropertiesBox item\_properties; // optional  
 ItemDataBox item\_data; // optional  
 GroupsListBox entity\_groups; // optional  
 Box other\_boxes[]; // optional  
}

## Discussion

It is possible to declare optional boxes in the ISOBMFF because:

1. Boxes can be disambiguated using the box\_type field in the header.
2. The box size in the box header allows the parser to determine if the end of the box is reached. If not, then this means that some additional boxes are present.

## Possible solutions

## Optional class members

In SDL, it can happen that a class may contain a given class based on the context it is in. In some environment, this class will be needed in some other this class would not be needed. As a result, the SDL author may want to declare a nested class instance to be optional.

For example:

class Foo  
{  
 unsigned int(8) a;  
 optional Bar bar;  
}

class Log

{

Foo foo;

}

In this class MyClass, the body of Foo does declare bar and but if bar is not in the parsed data, then the member foo.bar does not exist and thus would end up in an unspecified behaviour.

## Optional keyword

Classes are the mechanism with which declarations of composite types is performed. Their syntax is as follows.

Rule C.2: Class data types

[optional] class\_identifier *class\_variable\_*identifier**;**

The keyword optional indicates that the class instance may or may not be present. To use this keyword, the class shall be defined with a class identifier.

[Editor’s note: member class mark as typeid? Or should we say that class shall be a derived class of base class for which the class identifier is defined].

Accessing an optional member of class that was not parsed, and thus does not exist in the class instance, result in an unspecified behaviour. Consequently, the value of this class are also unspecified.

NOTE The SDL specification does not provide a mean to check whether the member of class exists.

# Topic #4: Array of boxes for ISOBMFF

## On container box

In this scenario, a box is meant to be a container for other boxes. For example, the movie box is defined as follows:

**Movie box**

**Definition**

Box Type: 'moov'  
Container: File  
Mandatory: Yes  
Quantity: Exactly one

The structure-data for a presentation is stored in the single MovieBox which occurs at the top-level of a file. Normally this box is close to the beginning or end of the file, though this is not required.

**Syntax**

aligned(8) class MovieBox extends Box('moov')  
{  
}

From the SDL declaration of the movie box, nothing is specifying what the box can contain. The semantic is also silent on what it contains. Instead, the philosophy of the ISOBMFF specification is to specify where a box can be located and not what it can contain.

Therefore, strictly speaking, the movie box as declared is an empty box.

Another example of a container box is the MetaBox. In this case, specification declare an array of element of the class Box.

aligned(8) class MetaBox (handler\_type)  
 extends FullBox('meta', version = 0, 0)   
{  
 …  
 Box other\_boxes[]; // optional  
}

Note that the comment mentions optional which would imply that the array may be empty which is currently a topic of discussion in the File Format group (regarding the element track\_IDs[])

More generally, this syntax of declaring a generic class would be possible is the class Box would be an abstract class and that all derive classes would use the SDL mechanism of extension ID. However, those conditions are not met the current ISOBMFF specification.

## Discussion

In ISOBMFF, there are several places with a box can contains any boxes.

aligned(8) class MetaBox (handler\_type)  
 extends FullBox('meta', version = 0, 0)   
{  
 …  
 Box other\_boxes[]; // optional  
}

The statement “Box other\_boxes[]” is actually possible in SDL but ISOBMFF doesn’t currently use the SDL polymorphism feature (class identifier in the class declaration) to use this feature.

There seems to be three cases of container box:

1. A box can contain any other boxes.
2. A box can contain zero or more boxes from a any boxes in any order.
3. A box can contain zero or more boxes from a predetermined list of boxes in a specific order.

For case 1), the SDL has some provision but requires using specific tools which limits the freedom of the SDL declaration author.

For case 2), the SDL does not seem to have any tools for that.

For case 3), this is almost enabled in SDL as illustrated in the meta box. The only missing feature is to declare that the boxes are optional.

## Possible solutions

## New keyword “class id”

Declaring a class identifier keyword to declare which class member in the body should be used as class identifier.

Possible keywords are:

* **typeid**
* **id**
* **class\_id**

EXAMPLE 1 ⎯

class Foo : bit(2) id = 0 {

// note that as "id" is declared it is accessible within this class

// as a constant value and lengthof(id) will return 2

int(5) a; // this a is preceded by the 2 bits of id

}

class Bar {  
 id bit(2) class\_type = 0;  
 int(5) a; // this a is preceded by the 2 bits of id

}

Those two classes Foo and Bar would be syntactically equivalent.

To extend from it, the regular derivation process can be used but the SDL author has to make sure the same class id is defined in each derived class and at the same bit position in the class.

class Foo1 extends Foo {

id bit(2) class\_type = 1;

int(3) b;

}

class Foo2 extends Foo {  
 id bit(2) class\_type = 2;

int(5) c;

}

## Extending polymorphism declaration

Currently, the polymorphism feature works by defining a field bit which will be written as the first element of the class.

The issue with this in ISOBMFF, and possibly other specifications, is that the box type is written in the second position after the size of the box.

Therefore, the current polymorphism could be extended to allow for declaring the bit offset from which to start reading the class id.

For example, one way would be to add an optional bit offset, when absent the offset is null.

class FooBox : 32+bit(32) type = 'moov' {

unsigned int(32) size;  
 bit(32) type;  
 if (size==1) {  
 unsigned int(64) largesize;  
 } else if (size==0) {  
 // box extends to end of file  
 }  
 if (boxtype=='uuid') {  
 unsigned int(8) usertype[16] = extended\_type;  
 }

…

}

Alternatively, we could also forbid the redeclaration in the class.

class FooBox : 32+bit(32) type = 'moov' {

unsigned int(32) size;  
 // Here is the bit(32) type field but it is already declared in the class declaration   
 if (size==1) {  
 unsigned int(64) largesize;  
 } else if (size==0) {  
 // box extends to end of file  
 }  
 if (boxtype=='uuid') {  
 unsigned int(8) usertype[16] = extended\_type;  
 }

…

}

# Topic #5: An unordered set of boxes

## General

Here the case is that a box contain a sequence of box wherein each box is optional. However, the order of which the boxes appear is not specified and any order is allowed.

We can call this case an unordered set of boxes where set is understood as the concept of unordered set as in C++:

std::unordered\_set is an associative container that contains a set of unique objects of type Key.

So those solutions provide mean to express this type of data.

In spirit, it is also similar to the XML element all:

From 3.8.4.1.3 All-groups , <https://www.w3.org/TR/xmlschema11-1/>

[Definition:]  A **grouping** of a sequence is a set of sub-sequences, some or all of which may be empty, such that each member of the original sequence appears once and only once in one of the sub-sequences and all members of all sub-sequences are in the original sequence.

## Possible solutions

## New syntax for “an unordered set / all XML element”

Let’s take this box as example and assume the order of the clap and pasp boxes can be any.

class VisualSampleEntry(codingname) extends SampleEntry (codingname)  
{  
 unsigned int(16) pre\_defined = 0;  
..   
 CleanApertureBox clap; // optional  
 PixelAspectRatioBox pasp; // optional  
}

One way of expressing the unordered set it.

class UnorderedBox {

all {

   Second b;

    First a;

  }

}

One parsing can lead to:

First

Second

Another parsing can lead to:

Second

First

To be more explicit, the syntax could also incorporate the way to disambiguate the set, for example:

class UnorderedBox {

all : bit(32) id {

   Second b;

    First a;

  }

}

This mean that the parser has to read id as the next 32 bits to identify it correspond to b.id or a.id.

Another alternative is to declare the base class before the unordered set.

class UnorderedBox {

all : Box {

   FooBox b;

    BarBox a;

  }

}

In this case, FooBox and BarBox derive from Box and per their definition, the parser knows how to disambiguate between a and b.

# Topic #6: Import directive

## Motivation

We can envision two ways to reuse SDL definitions from other files, similar to include foo.xxx and import foo behaviour commonly used in programming language.

With include, the targeted file is simply replaced by the directive in the current file. The feature is simple to interpret. However, it typically requires to handle duplicated declarations when the same file is included multiple files, e.g. #define guard in C/C++.

Alternatively, the import functionality typically refer to importing the function of loading the declaration found into the targeted file in the current scope of the document. It thus does not imply the replacement by the targeted file itself.

As a result, it appears less to define an import feature instead of an include function which has to come with guard features.

## Text proposal

**Directives**

The following directives are defined.

Rule D.1: import

import path/to/module

This directive imports the SDL declarations found in the file called module.sdl at the located at the path “path/to”. After the import, all the classes defined in this imported module can be used in the subsequent SDL declarations.

# Topic #7: SDL source file definition

A SDL file is a plain text file with extension .sdl which contains a set of class object declarations and comment lines. Comment lines are defined as in 4.2 Comments. The character encoding could also be considered.

The Annex A which contains the SDL grammar is actually already anticipating a file containing the SDL declaration.

# Topic #8: Unicode inconsistencies

## Motivation

Clause 5.1 permits use of UCS characters "except" (a different way of saying "shall not"?) certain characters:

-          "the BOM character (0xFEFF)"

-          "non-printable characters (0x00 to 0x1F inclusive)"

-          "characters causing line feeds (0x85, 0x2028, 0x2029"

## Comment #1

First, a comment about citation of UCS characters: the notation "0x" is a convention from C or similar specifications for referring to hexadecimal integer values in a machine representation. It is not a convention used in ISO/IEC 10646 to refer to characters or code points. Related, it's important to note that UCS code points are integer values in the range 0000 to 10FFFF and conceptually distinct from any machine representation. When referring to 10646, the conventions of 10646 should be used. The relevant conventions are defined in clause 7.6, "Short identifiers for code points (UIDs)". Quoting a summary statement,

The full syntax of the notation of a short identifier, in Backus-Naur form, is   
                { U | u } {+}(xxxx | xxxxx | xxxxxx)

where "x" represents one hexadecimal digit (0 to 9, A to F, or a to f).

Thus "0x00 to 0x1F" should be cited as "U+0000 to U+001F"; "0x85, 0x2028, 0x2029" as "U+0085, U+2028, U+2029". (The "U+" is optional, but in this document should be included for clarity.)

Note: on this point, this document could do well to follow its own requirement for string literals in 5.18:

"A universal character code point shall be defined using 4 uppercase hexadecimal characters prefixed with **\u** or 8 uppercase hexadecimal characters prefixed with **\U**"

Also, btw, the distinction I mention between code points and machine representations is very important in 10646 and Unicode. See [UTR #17](https://www.unicode.org/reports/tr17/" \o "https://www.unicode.org/reports/tr17/) and [RFC2130](https://www.ietf.org/rfc/rfc2130.txt" \o "https://www.ietf.org/rfc/rfc2130.txt) for background. I'll touch on this again below.

## Comment #2

10646 does not anywhere refer to "BOM". The character assigned to code point U+FEFF is named ZERO WIDTH NO-BREAK SPACE. The notion of a byte order "signature" is described in clause 7.3.7, but the terminology "BOM" or "byte order mark" is not used in 10646; rather, it comes from The Unicode Standard (see [Unicode 16.0, 2.13.2](https://www.unicode.org/versions/Unicode16.0.0/core-spec/chapter-2/" \l "G9354" \o "https://www.unicode.org/versions/Unicode16.0.0/core-spec/chapter-2/#G9354)).

## Comment #3

Clause 5.1 states that an SDL specification cannot contain "BOM", but 6.6 states contexts in which BOM will be required. IIUC, this all seems to imply there can be cases in which the SDL description would need to specify a string literal that does include BOM, and by 5.1 and 5.18 it would need to be cited within a string literal using an escaped representation such as u"\ufeff". That seems to be worth mentioning in 5.1 in a note.

## Comment #4

Re this statement in 5.1:

The character set and encoding used to store an SDL specification in a file is not specified in this document. A computer program implementation or a standard containing an SDL specification may choose to specify such details.

This statement seems to be confused about the meaning of "character set". As noted in [RFC2130](https://www.ietf.org/rfc/rfc2130.txt" \o "https://www.ietf.org/rfc/rfc2130.txt), "character set" has been used with different meanings in different contexts. That document uses "Coded Character Set" unambiguously in a manner consistent with ISO/IEC 10646 and Unicode. This document should do likewise.

Since this document is referencing 10646, then "character set" should be understood in the nearest terminology of that standard, namely "Coded Character Set", in which case it is incorrect to state that this document doesn't specify a "character set": it does precisely that in clause 5.1 by specifying "the basic character set" and by reference to 10646, "Universal Coded Character Set".

## Comment #5

Regarding encoding, it's appropriate to state in 5.1 that the encoding for *an SDL specification in a file* is not specified. But since an SDL description is specifying the format and interpretation of data bit streams, the encoding of strings is critical. Clause 6.6 addresses this.

However, I think the options in 6.6 for UCS encoding forms and encoding schemes could be a bit too limited. In particular, it does not permit use of the UTF-32 encoding form, and requires the byte order of UTF-16 encoding schemes to be disambiguated using BOM rather than being able to specify the encoding scheme directly in the SDL description. For instance, if one wanted to apply SDL to ISO/IEC 14496-22, then that would be a problem since that specifies that certain strings shall be represented in the UTF-16BE encoding scheme (cf. 11.3 of ISO/IEC 10646).

## Comment #6

Because this document is referring to UCS characters by numeric values (I'll assume code points are intended, not machine representations), 5.1 restricts far fewer characters than it probably should, and a lot of ambiguity is left.

1. 10646 specifies a number of non-character code points. They probably should not be permitted in an SDL description or in string values, but this document does not restrict these.

b) 10646 reserves code points D800 to DFFF for use in the UTF-16 encoding form. While it doesn't describe them as such, these are effectively non-character code points - that is, those code points will never be assigned to characters. Those \_*code points*\_ should not be referred to in an SDL description or in string literals, but this document does not restrict these.

c) Clause 5.1 cites U+0000 to U+001F as restricted "non-printable characters", but UCS contains many other non-printable characters.

* Other JTC1/SC2 standards specified "C0" and "C1" control characters, and 0000 to 001F correspond to the C0 control characters only. The C1 controls are encoded as 0080 to 009F; clause 5.1 excludes 0085, but all of 0080 to 009F should be excluded from SDL descriptions. Also, 007F is also classified in UCS as a control character and should be excluded from SDL descriptions.
* UCS also has 170 format control characters characters - see [this query](https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3Agc%3DCf%3A%5D&g=&i=" \o "https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3Agc%3DCf%3A%5D&g=&i=) for the complete set. Some of these can potentially be problematic in an SDL description.   
    
  In fact, some of these format controls can be \*\*dangerous\*\* in an SDL description if it is intended to be machine-readable; this is because the interpretation of the file by a human reviewer could be significantly different from the interpretation by a machine unless the tool used by the human reviewer implements particular security mitigations. See [Unicode Technical Standard #55,](https://www.unicode.org/reports/tr55/" \o "https://www.unicode.org/reports/tr55/) *[Unicode Source Code Handling](https://www.unicode.org/reports/tr55/" \o "https://www.unicode.org/reports/tr55/)* for details.
* UCS characters U+FFFC OBJECT REPLACEMENT CHARACTER and U+FFFD REPLACEMENT CHARACTER are not classified as control characters, but they should not be permitted in an SDL description: there's no useful scenario for them in an SDL description, and their use could only be confusing.
* UCS also includes other layout control characters that are not classified as control characters per se, but that are used to control visual presentation in certain situations and that, on their own, are not visible. At least the following probably should not be permitted in an SDL description:
  + U+00AD SOFT HYPHEN
  + U+034F COMBINING GRAPHEME JOINER
  + U+200B ZERO WIDTH SPACE
  + U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER
  + U+FE00..U+FE0F, U+E0100 VARIATION SELECTOR-1 .. U+E01EF VARIATION SELECTOR-256
  + Tag characters U+E0020 TAG SPACE .. U+E007F CANCEL TAG

[This query](https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3ADefault_Ignorable_Code_Point%3A%5D&g=&i=" \o "https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3ADefault_Ignorable_Code_Point%3A%5D&g=&i=) shows code points that are classified as Default Ignorable; any of the assigned characters are non-printing at least in isolation if not in all contexts.

## Comment #7

Clause 5.5 states that identifiers "may be comprised of ... Latin alphabetic characters". This clause doesn't mention that the set so-defined is not a stable set, but can grow and has grown over time with new UCS versions.

[This query](https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3Asc%3DLatn%3A%5D%26%5B%3Agc%3DL%3A%5D&g=age&i=gc" \o "https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3Asc%3DLatn%3A%5D%26%5B%3Agc%3DL%3A%5D&g=age&i=gc) lists all 1,448 Latin alphabetic characters grouped by age (Unicode version), showing how that set has grown over time. If an implementation is parsing an SDL description in a machine-readable file, then it could need to know whether it should expect a stable identifier space. [Unicode Standard Annes #31,](https://www.unicode.org/reports/tr31/" \o "https://www.unicode.org/reports/tr31/) *[Unicode Identifiers and Syntax](https://www.unicode.org/reports/tr31/" \o "https://www.unicode.org/reports/tr31/)*, discusses such issues (summarized in the Introduction). If the intent that identifiers can only use the Latin letters specified as part of the "basic" character set defined in clause 5.1, then it should be explicit about that; but if identifiers can use other Latin characters in the UCS, then it should address the issue of stability.

That clause also states that identifiers may include "digits" but isn't explicit by what that means. Clause 5.1 lists "0 .. 9" as part of the "basic" character set; if that's all that's intended, then that should be stated explicitly. Otherwise, [this query](https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3Agc%3DNd%3A%5D&g=&i=gc" \o "https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%3Agc%3DNd%3A%5D&g=&i=gc) shows all UCS characters classified as digits. (Btw, that set can also grow over time.)