ISO Document: ISO/IEC ——:2024(E)
Document Number: P3342R1
Date: 2024-11-21
Revises:
Reply to: ISO C++ Tooling Study Group
sg15@lists.isocpp.org

Working Draft, Standard for C++ Ecosystem

Note: this is an early draft. It's known to be incomplet and incorrekt, and it has lots of bad formatting.

Contents

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see `https://www.iso.org/directives`).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see `https://www.iso.org/patents`).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) (see `https://www.iso.org/iso/foreword.html`).

This document was prepared by Technical Committee ISO/IEC JTC1, Information technology, Subcommittee 22, Programming languages, their environments and system software interfaces, Working Group 21, C++.

A list of all parts in the ISO/IEC —— series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at `https://www.iso.org/members.html`.

The main changes are as follows:

— Initial release.

1 Scope [intro.scope]

1 This document specifies formats, processes, definitions, and so on, that facilitates the interoperation of the tools and systems that implement, and interface with, the C++ programming language.

2 C++ is a general purpose programming language described in ISO/IEC 14882:2020 *Programming languages — C++* (hereinafter referred to as the *C++ standard*).

2 Normative references [intro.refs]

1 The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

Vocabulary ISO/IEC 2382, *Information technology — Vocabulary*

POSIX ISO/IEC 9945:2009, *Information technology — Portable Operating System Interface (POSIX®) Base Specifications, Issue 7*

C++ ISO/IEC 14882:2020, *Programming languages — C++*

JSON ISO/IEC 21778:2017, *Information technology — The JSON data interchange syntax*

Mathematics ISO 80000-2:2019, *Quantities and units — Part 2: Mathematics*

SemVer The SemVer Team. *Semantic Versioning 2.0.0.* June 18 2013. Available at: `https://semver.org/spec/v2.0.0.html`

3 Conformance [intro.cnf]

1 A conforming implementation shall meet the following criteria for conformance to this standard:

(1.1) — An application shall support the minimum level functionality of introspection (5.9).

4 Terms and definitions [intro.defs]

1 For the purposes of this document, the terms and definitions given in ISO/IEC 2382, the terms and definitions given in ISO/IEC 14882:2020, and the following apply.

2 ISO and IEC maintain terminology databases for use in standardization at the following addresses:

(2.1) — ISO Online browsing platform: available at `https://www.iso.org/obp/`

(2.2) — IEC Electropedia: available at `https://www.electropedia.org/`

3 Terms that are used only in a small portion of this document are defined where they are used and italicized where they are defined.

4.1 [defns.application]
application
a computer program that performs some desired function.

[*Note 1*: From POSIX. — *end note*]

4.2 [defns.capability]
capability
an aspect of an overall specification that defines a subset of the entire specification.

4.3 [defns.directory]
directory
a file that contains directory entries.

[*Note 1*: From POSIX. — *end note*]

4.4 [defns.direntry]
directory entry
an object that associates a filename with a file.

[*Note 1*: From POSIX. — *end note*]

4.5 [defns.file]
file
an object that can be written to, or read from, or both.

[*Note 1*: From POSIX. — *end note*]

4.6 [defns.filename]
filename
a sequence of bytes used to name a file.

[*Note 1*: From POSIX. — *end note*]

4.7 [defns.parentdir]
parent directory
a directory containing a directory entry for the file under discussion.

[*Note 1*: From POSIX. — *end note*]

4.8 [defns.pathname]
pathname
a string that is used to identify a file.

[*Note 1*: From POSIX. — *end note*]

5 Introspection [intspct]

5.1 Preamble [intspct.pre]

1 This clause describes options, output, and formats that describe what capabilities of this standard an application supports. An application shall support the *minimum level* functionality (5.9). An application can support the *full level* functionality (5.10).

2 This clause specifies the `std.info` capability (5.7).

5.2 Overview [intspct.overview]

1
```
application [ std-info-opt [ declaration ] ] [ std-info-out-opt file ]
```

5.3 Options [intspct.options]

1 Applications shall accept one of two options syntax variations: `--name=value` (`--name` without a value) or `-name:value` (`-name` without a value).

2 Applications shall indicate an error if invoked with an option syntax variation that it does not support.

[*Note 1*: An application will report the error in what is conventional for the platform it runs in. On POSIX and Windows it would return an error code, and optionally output to the error stream. — *end note*]

[*Note 2*: It is up to a program that interacts with an application implementing introspection to determine what option syntax variation the application supports. One method to accomplish that is to execute the application with one of the two syntax styles and use the error indication to conclude which syntax works. Another is to have a priori knowledge of which syntax variation works. — *end note*]

5.3.1 Information Option [intspct.opt.info]

1 This option shall be supported.

2 *std-info-opt*

3 Outputs the version information of the capabilities supported by the application. The option is specified as `--std-info` or `-std-info`. The option can be specified zero or one time. The application shall support the option for *minimum level* (5.9) functionality.

5.3.2 Information Output Option [intspct.opt.out]

1 This option shall be supported.

2 *std-info-out-opt file*

3 The pathname of a file to output the information to. The option is specified as `--std-info-out=file` or `-std-info-out:file`. If *file* is '`-`', the standard output shall be used. The application shall support the option for *minimum level* (5.9) functionality. Not specifying this option while specifying the *std-info-opt* option (5.3.1) shall be equivalent to also specifying a *std-info-out-opt file* option where *file* is '`-`'.

5.3.3 Declaration Option [intspct.opt.decl]

1 This option should be supported.

2 *std-info-opt declaration*

3 Declares the required capability version of the application. The option is specified as `--std-info=`*declaration* or `-std-info:`*declaration*. The option can be specified any number of times. The application shall support the option for *full level* (5.10) functionality.

5.4 Output [intspct.output]

1 An application shall output a valid JSON text file that conforms to the introspection schema (5.6) to the file specified in the options (5.3.2).

5.5 Files [intspct.file]

1 An application can provide an *introspection file* that contains valid JSON that conforms to the introspection schema (5.6).

2 An *introspection file* shall contain the same information as that produced from the *std-info-opt* information option (5.3.1).

3 An *introspection file* shall be named the same as the application with any filename extension replaced with the `stdinfo` filename extension. It is implementation defined how the filename of the introspection file replaces the application filename extension with the new `stdinfo` filename extension.

[*Note 1*: For Windows, POSIX, and other platforms replacing the filename extension would remove any filename bytes after the last period (U+002E FULL STOP) and append the `stdinfo` sequence of bytes. — *end note*]

4 An *introspection file* shall either have the same parent directory as the application, have an implementation defined parent directory that is relative to the parent directory of the application, or have an implementation defined parent directory.

5.6 Schema [intspct.schema]

1 An introspection JSON text file shall contain one introspection JSON object (5.6.1).

5.6.1 Introspection Object [intspct.schema.obj]

1 The *introspection object* is the root JSON object of the introspection JSON text.

2 An *introspection object* can have the following fields.

5.6.2 JSON Schema Field [intspct.schema.schema]

1 *Name*: `$schema`

2 *Type*: `string`

3 *Value*: The value shall be a reference to a JSON Schema specification.

4 *Description*: An *introspection object* can contain this field. If an *introspection object* does not contain this field the value shall be a reference to the JSON Schema corresponding to the current edition of this standard.

5.6.3 Capability Field [intspct.schema.cap]

1 *Name*: *capability-identifier* (5.7)

2 *Type*: `string` or `array`

3 *Value*: (for `string`) The value shall be a *version-number* for *minimum level* functionality. Or the value shall be a *version-range* for *full level* functionality.

4 *Value*: (for `array`) The value can be a JSON `array` for *full level* functionality. If the value is a JSON `array` the items in the `array` shall be a *version-number* or *version-range*.

5 *Description*: An *introspection object* can contain this field one or more times. When the field appears more than one time the name of the fields shall be unique within the *introspection object*.

5.7 Capabilities [intspct.cap]

capability-identifier:
  name scope-designator name sub-capability-identifier$_{opt}$

sub-capability-identifier:
  scope-designator name sub-capability-identifier$_{opt}$

name:
  one or more of:
  U+*0061* .. U+*007A* LATIN SMALL LETTER A .. Z
  U+*005F* LOW LINE

scope-designator:
  U+*002E* FULL STOP

1 A *capability-identifier* is composed of two or more *scope-designator* delimited *name* parts.

2 The *name* `std` in a *capability-identifier* is reserved for capabilities defined in this standard.

3 Applications can specify vendor designated *name* parts defined outside of this standard.

5.8 Versions [intspct.vers]

1 A version shall be either a single version number (5.8.1) or a version range (5.8.2).

2 A single version number shall be equivalent to the inclusive version range spanning solely that single version number.

[*Note 1*: That is, the version number `i.j.k` is equivalent to version range `[i.j.k,i.j.k]`. — *end note*]

5.8.1 Version Number [intspct.vers.num]

1 A version number shall conform to the SemVer `<version core>` syntax.

2 A version number can be truncated to only `<major>` or `<major>.<minor>` syntax.

3 A version number composed of only `<major>` is equivalent to `<major>.0.0`.

4 A version number composed of only `<major>.<minor>` is equivalent to `<major>.<minor>.0`.

5 Version numbers define a total ordering where version number a is ordered before a version number b when a has a lower SemVer precedence than b.

5.8.2 Version Range [intspct.vers.range]

version-range:
  version-range-min-bracket
  version-min-number version-range-max-part$_{opt}$
  version-range-max-bracket

version-range-max-part:
  U+002C COMMA *version-max-number*

version-min-number:
  version-number

version-max-number:
  version-number

version-range-min-bracket:
  one of:
  U+005B LEFT SQUARE BRACKET
  U+0028 LEFT PARENTHESIS

version-range-max-bracket:
  one of:
  U+005D RIGHT SQUARE BRACKET
  U+0029 RIGHT PARENTHESIS

1 A version range is composed of either one version number bracketed, or two version numbers separated by a U+002C COMMA and bracketed.

[*Example 1*: A version range with a single version number "`[1.0.0]`". — *end example*]

[*Example 2*: A version range with a two version numbers "`[1.0.0,2.0.0]`". — *end example*]

2 A version range a that is `[`i`,`j`]` makes i and j inclusive version range numbers, defining a Mathematics closed interval.

3 A version range a that is `(`i`,`j`)` makes i and j exclusive version range numbers, defining a Mathematics open interval.

4 A version range a that is `(`i`,`j`]` makes i an exclusive version number and j an inclusive version number, defining a Mathematics half-open interval.

5 A version range a that is `[`i`,`j`)` makes j an exclusive version number.

6 A version range with a single inclusive version number x is equivalent to the version range `[`x`,`x`]`.

7 A version range with a single exclusive version number x is invalid.

8 An exclusive version number x does not include the version number x when compared to another version number y.

9 A version range a with version numbers i and j when compared to a version range b with version number m and n will result in an empty version range when: $j < m$ or $n < i$.

10 Otherwise if i or m are inclusive version numbers and if j or n are inclusive version numbers the resulting range when a is compare to b is the inclusive version numbers "lesser of i and m" and "lesser of j and n".

11 Otherwise if i or m are inclusive version numbers and if j or n are inclusive version numbers the resulting range when a is compare to b is the inclusive version number "lesser of i and m" and the exclusive version number "lesser of j and n".

12 Otherwise if j or n are inclusive version numbers the resulting range when a is compared to b is the exclusive version number "lesser of i and m" and the inclusive version number "lesser of j and n".

13 Otherwise the resulting range when a is compared to b is the exclusive version numbers "lesser of i and m" and "lesser of j and n".

5.9 Minimum Level [intspct.min]

1 An application that supports the *minimum level* functionality indicates it by specifying a single version (5.8.1) as the value of the `std.info` capability (5.7).

[*Example 1*: `{ "std.info": "1.0.0" }` — *end example*]

5.10 Full Level [intspct.full]

1 An application can support the *full level* functionality as defined in this section. An application that reports supporting the *full level* functionality shall support all of the functionality in this section.

2 An application that supports the *full level* functionality indicates it by specifying a version range (5.8.2) or an array of version range items as the value of the `std.info` capability (5.7).

[*Example 1*: `{ "std.info": "[1.0.0]" }` — *end example*]

3 An application that responds with an array of version range items as the value of a capability field shall support the union of the range items indicated.

5.11 Introspection Information [intspct.info]

1 An application shall output an introspection schema (5.6) that contains one capability field for each capability that the application supports when given the `--std-info` option (5.3.1).

2 An application shall indicate the single version (5.8.1) or version range (5.8.2) of each capability it supports as the value of the capability field.

5.12 Introspection Declaration [intspct.dcl]

1 An application that supports the *full level* functionality when given one or more *std-info-opt declaration* options shall conform its functionality to the indicated edition of this standard in the given *declaration version-number* for the given capability.

declaration:
    capability-identifier U+003D EQUALS SIGN *version-number*

2 An application, when not given a *std-info-opt declaration* option for a capability it supports, should conform its functionality to the most recent version of the standard it supports for that capability.

3 An application, when given a capability declaration option and the given version is outside of the version range that the application supports, should indicate an error.

5.13 Compatability [intspct.compat]

1 An application shall indicate, per SemVer specification, that version n of the interface it implements is *backward compatible* with another version p of the interface that another application implements when the `<major>` number is the same in version n and p and version n follows version p.

6 Structured Parameters [strctparam]

6.1 Preamble [strctparam.pre]

1 This clause describes options, output, and formats that control the behavior of applications through the specification of arguments and options from *structured parameters*.

2 This clause specifies the `std.strctparam` capability (5.7) version `1.0.0`.

3 An application can implement this capability.

4 An application that implements the `std.strctparam` capability shall include the `std.strctparam` field and version value in the introspection JSON text output (5.6.3).

6.2 Overview [strctparam.overview]

1 *`application [ std-strctparam-input file ]`*

6.3 Input Option [strctparam.input]

1 *std-strctparam-input*

2 The pathname of a file to read the *structured parameters* from. The option is specified as `--std-param=`*file* or `-std-param:`*file*. If *file* is '`-`', the standard input shall be used.

6.4 Files [strctparam.file]

1 An application shall read a valid JSON text file that conforms to the *structured parameters* schema (6.5).

2 An application shall interpret the information in the file as if the options and arguments in the file occur in the same position as the *std-strctparam-input* parameter of the list of parameters given to the application. Given either directly as part of the application command line or as part of the arguments field (6.5.4).

3 An application shall process arguments in the file in the order that they appear.

4 An application shall process options in the file as if the a new modified state replaces the current state.

6.5 Schema [strctparam.schema]

1 A *structured parameters* JSON text file shall contain one *structured parameters* JSON object (6.5.1).

6.5.1 Structured Parameters Object [strctparam.schema.obj]

1 The *structured parameters object* is the root JSON object of the structured parameters JSON text.

2 A *structured parameters object* can have the following fields.

3 A *structured parameters object* shall have only one of the `arguments` and `options` fields.

6.5.2 JSON Schema Field [strctparam.schema.schema]

1 *Name*: `$schema`

2 *Type*: `string`

3 *Value*: The value shall be a reference to a JSON Schema specification.

4 *Description*: A *structured parameters object* can contain this field. If a *structured parameters object* does not contain this field the value shall be a reference to the JSON Schema corresponding to the current edition of this standard (Annex B).

6.5.3 Version Field [strctparam.schema.ver]

1 *Name*: `version`

2 *Type*: `string`

3 *Value*: `1` or `1.0` or `1.0.0`

4 *Description*: The version field indicates the version of the *structured parameters* represented in the contents of the JSON text. If a *structured parameters object* does not contain this field the value shall be `1.0.0`.

6.5.4 Arguments Field [strctparam.schema.args]

1 *Name*: `arguments`

2 *Type*: `array`

3 *Value*: The value shall be a JSON `array`. The items in the `array` shall be of JSON `string` types.

4 *Description*: The arguments field specifies items to be interpreted directly as if they occur in the command line of the program.

5 The application shall process the items as if they replace the `\emph{std-strctparam-input}` argument.

6.5.5 Options Field [strctparam.schema.opts]

1 *Name*: `options`

2 *Type*: `object`

3 *Value*: The value shall be a JSON `object`.

4 *Description*: A *structured parameters object* can contain this *structured options* field. The keys of items shall be *name* (6.5.6) values.

6.5.6 Names [strctparam.schema.names]

name:
  one or more of:
  U+0061 .. U+007A LATIN SMALL LETTER A .. Z
  U+0030 .. U+0039 DIGIT ZERO .. NINE
  U+005F LOW LINE U+002D HYPHEN-MINUS

scope:
  name scope-designator $scope_{opt}$

scope-designator:
  U+002E FULL STOP

1 The *name* `std` is reserved for *structured options* defined in this standard.

2 Applications can specify vendor designated *name* parts outside of this standard.

6.5.7 Structured Option std.param [strctparam.schema.opt.param]

1 The `std.param` *structured option* defines an option to refer to additional *structured parameters* (Clause 6) to process.

2 An application shall implement this option.

3 The `std.param` option shall have the following fields.

4 *Name*: `pre`

5 *Type*: `string` or `array`

6 *Value*: (for `string`) A pathname to a file containing *structured parameters* (Clause 6).

7 *Value*: (for `array`) A list of pathname `string` items to files containing *structured parameters* (Clause 6)

8 *Description*: One or more references to files that include additional *structured parameters* (Clause 6).

9 *Name*: `post`

10 *Type*: `string` or `array`

11 *Value*: (for `string`) A pathname to a file containing *structured parameters* (Clause 6).

12 *Value*: (for `array`) A list of pathname `string` items to files containing *structured parameters* (Clause 6)

13 *Description*: One or more references to files that include additional *structured parameters* (Clause 6).

14 A `string` value in the `pre` or `post` field shall be as if the value was given as an `array` with the `string` value as the only value in the `array`.

15 An application shall process the *structured parameters* (Clause 6) in the `pre` field before processing the *structured options* where the `std.param` option is specified.

16 An application shall process the *structured parameters* (Clause 6) in the `post` field after processing the *structured options* where the `std.param` option is specified.

17 An application shall process the *structured parameters* (Clause 6) in the `pre` and `post` field in the order given in the value applying semantics as specified in the parameters or options of the *structured parameters*.

7 Structured Core Options [strctopt.core]

7.1 Preamble [strctopt.core.pre]

1 This clause describes the schema and semantics of core *structured options* (6.5.5) for C++ compiler front-end applications.

2 This clause specifies the `std.strctopt.core` capability (5.7) version `1.0.0`.

3 An application can implement this capability.

4 An application that implements the `std.strctopt.core` capability shall include the `std.strctopt.core` field and version value in the introspection JSON text output (5.6.3).

7.2 Source [strctopt.core.src]

1 *Name*: `source`

2 *Type*: `array`

3 *Value*: The value shall be a JSON `array`. The items in the `array` shall be of JSON `object` items specifying source objects (7.2.1).

4 *Description*: Defines a list of source files for an application to process.

5 The application shall add the given sources to the set of files to process.

6 The sources given shall be appended to existing sources in the order given.

7.2.1 Source Object [strctopt.core.srcobj]

1 A source object shall have a `name` field.

2 *Name*: `name`

3 *Type*: `string`

4 *Value*: A pathname to a file.

5 *Description*: The pathname of the source file.

6 A source object can have a language field (7.10).

7 Specifying a language field for a source shall replace any other determination of the source language by the application.

8 Specifying a language field for a source shall replace a language field specified in the *structured parameters* options field (6.5.5).

9 A source object can have a kind field (7.11).

10 Specifying a kind field for a source shall replace any other determination of the source kind by the application.

11 Specifying a kind field for a source shall replace a kind field specified in the *structured parameters* options field (6.5.5).

12 A source object can have a vendor field (7.9).

7.3 Output [strctopt.core.out]

1 *Name*: `output`

2 *Type*: `array`

3 *Value*: The value shall be a JSON `array`. The items in the `array` shall be JSON `object` items specifying output objects (7.3.1).

4 *Description*: Specifies the output files to generated when processing sources (7.2).

7.3.1 Output Object [strctopt.core.outobj]

1 An output object shall have a `name` field.

2 *Name*: `name`

3 *Type*: `string`

4 *Value*: A pathname to a file.

5 *Description*: The name of the output file.

6 An output object can have a kind field (7.11).

7 Specifying a kind field for output shall replace any other determination of the output kind by the application.

8 An output object can have a vendor field (7.9).

7.4 Include Directories [strctopt.core.incd]

1 *Name*: `include_dirs`

2 *Type*: `array`

3 *Value*: The value shall be a JSON `array`. The items in the `array` shall be JSON `string` items of pathnames to directories.

4 *Description*: One or more entries to directories that are searched by the header inclusion of a C++ preprocessor.

5 The application shall add the given directories to the set of directories searched by the header inclusion of a C++ preprocessor.

6 The directories given shall be appended to the existing include search directories in the order given.

7.5 Library Directories [strctopt.core.libd]

1 *Name*: `library_dirs`

2 *Type*: `array`

3 *Value*: The value shall be a JSON `array`. The items in the `array` shall be JSON `string` items of pathnames to directories.

4 *Description*: One or more entries to directories that are searched for libraries.

5 The application shall add the given directories to the set of directories searched for libraries.

6 The directories given shall be appended to the existing library search directories in the order given.

7.6 Define Preprocessor Symbols [strctopt.core.def]

1 *Name*: `define`

2 *Type*: `array`

3 *Value*: The value shall be a JSON `array`. The items in the `array` shall be JSON `object` items specifying preprocessor symbol definition objects (7.6.1).

4 *Description*: Specifies preprocessor symbols to define during processing of sources.

7.6.1 Preprocessor Symbol Definition Object [strctopt.core.def.sym]

1 *Name*: `name`

2 *Type*: `string`

3 *Value*: A valid preprocessor symbol.

4 *Description*: The field specifies the preprocessor symbol to define.

5 The symbol shall be valid for the consuming application.

6 The application shall indicate an error for invalid symbols.

7 *Name*: `value`

8 *Type*: `number`, `string`, `boolean`, or `null`

9 *Value*: A valid preprocessor define value.

10 *Description*: The field specifies the value to assign to the preprocessor symbol.

11 The value shall be valid for the consuming application.

12 The value can be omitted. When the value is omitted it shall be converted to a string value of `1`.

13 A `number` value shall be converted to a string value.

14 A `string` value shall be used as given.

15 A `true` value shall be converted to a string value of `1`.

16 A `false` value shall be converted to a string value of `0`.

17 A `null` value shall be converted to a string value of `1`.

7.7 Undefine Preprocessor Symbols [strctopt.core.undef]

1 *Name*: `undef`

2 *Type*: `array`

3 *Value*: The value shall be a JSON `array`. The items in the `array` shall be of JSON `string` defining preprocessor symbol names.

4 *Description*: Specifies preprocessor symbols to "undefine".

5 The symbols shall be valid for the consuming application.

6 The application shall indicate an error for invalid symbols.

7 The application shall evaluate this option after any `define` (7.6) options.

7.8 Optimization [strctopt.core.opt]

1 *Name*: `optimization`

2 *Type*: `string`

3 *Value*: An optimization object item (7.8.1)

4 *Description*: The optimization to apply when generating the output.

5 The application shall replace each existing field in the optimization object (7.8.1).

7.8.1 Optimization Object [strctopt.core.optobj]

1 An optimization object can have any of `compile` and `link` fields.

2 *Name*: `compile`

3 *Type*: `string`

4 *Value*: `off`, `minimal`, `speed`, `space`, or `debug`

5 *Description*: The amount or type of optimization to apply to the generated output.

6 An application shall not perform optimization when given the `off` value.

7 For `minimal`, `speed`, `space`, and `debug` values the application behavior is unspecified.

8 *Name*: `link`

9 *Type*: `boolean`

10 *Value*: `true` or `false`

11 *Description*: Specify if optimizations that happen for linked output generation happen.

12 An application shall not perform optimizations for linked output generation when the value is `false`.

13 For a `true` value the application behavior is unspecified.

14 An optimization object can have a vendor field (7.9).

7.9 Vendor [strctopt.core.vendor]

1 *Name*: `vendor`

2 *Type*: `object`

3 *Value*: A vendor object (7.9.1)

4 *Description*: Specifies vendor defined options to apply in the context they appear.

5 An application shall apply the vendor option semantics in the context they appear.

7.9.1 Vendor Object [strctopt.core.vendorobj]

1 A vendor object can have any number of fields.

2 The name of a field is unspecified.

3 The value of a field is unspecified.

[*Note 1*: It is up to application vendors to agree on the name fields. — *end note*]

[*Note 2*: It is up to application vendors to document the schema of the field values. — *end note*]

7.10 Language [strctopt.core.lang]

1 *Name*: `language`

2 *Type*: `object`

3 *Value*: A language object (??)

4 *Description*: The language to interpret the source as.

7.10.1 Language Object [strctopt.core.langobj]

1 A language object shall have a `name` field.

2 *Name*: `name`

3 *Type*: `string`

4 *Value*: One of: `c++` or an application defined value.

5 *Description*: Specifies the source text language.

6 Only a value of `c++` specifies that source text (7.2) is C++ ISO language.

7.11 Kind [strctopt.core.kind]

1 *Name*: `kind`

2 *Type*: `string`

3 *Value*: A kind item (7.11)

4 *Description*: The kind, or format, of the source file corresponding to the output file (7.3).

7.11.1 Kind Object [strctopt.core.kindobj]

1 *Name*: `name`

2 *Type*: `string`

3 *Value*: One of: `text`, `exec`, `object`, `dynamic_lib`, `archive_lib`

4 *Description*: The kind, or format, of source and output files.

5 A `text` value specifies that the source or output is textual.

6 A `exec` value specifies that the source or output is an executable program.

7 A `object` value specifies that the source or output is a linkable object.

8 A `dynamic_lib` value specifies that the source or output is dynamically linkable.

9 A `archive_lib` value specifies that the source or output is an archive, or collection, of linkable objects.

8 Diagnostics Output [diag]

8.1 Preamble [diag.pre]

1 This clause describes options, output, and formats that define formatted reporting of application diagnostics messages.

Annex A (informative)

Tool Introspection JSON Schema [intsjschm]

A.1 General [intsjschm.general]

1 This Annex defines the introspection capability schema (5.6) in terms of a *JSON Schema* (1.).

2 This JSON Schema can be referenced as the `$schema` field with URI value of `"std_info-1.0.0.json"`.

A.2 JSON Schema Specification [intsjschm.spec]

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "std_info-1.0.0.json",
  "title": "Tool Introspection Version 1.0.0 JSON Schema",
  "$defs": {
    "VersionMin": {
      "type": "string",
      "pattern": "^[0-9]+([.][0-9]+){0,2}$"
    },
    "VersionFull": {
      "type": "string",
      "pattern": "^[[(][0-9]+([.][0-9]+){0,2}[)\\]]$"
    },
    "VersionRange": {
      "type": "string",
      "pattern": "^[[(][0-9]+([.][0-9]+){0,2},[0-9]+([.][0-9]+){0,2}[)\\]]$"
    },
    "Version": {
      "oneOf": [
        {
          "$ref": "#/$defs/VersionMin"
        },
        {
          "$ref": "#/$defs/VersionFull"
        },
        {
          "$ref": "#/$defs/VersionRange"
        }
      ]
    },
    "Versions": {
      "type": "array",
      "items": {
        "$ref": "#/$defs/Version"
      }
    },
    "VersionSpec": {
      "oneOf": [
        {
          "$ref": "#/$defs/Version"
        },
        {
          "$ref": "#/$defs/Versions"
        }
      ]
    }
  },
  "anyOf": [
    {
      "type": "object",
      "properties": {
        "$schema": {
          "description": "JSON Schema URI for the version of the tool introspection format.",
          "type": "string",
          "format": "uri"
        },
        "std.info": {
          "description": "The Tool Introspection format version.",
          "$ref": "#/$defs/VersionSpec"
        }
      },
      "patternProperties": {
        "^[a-z_]+([.][a-z_]+)+$": {
          "$ref": "#/$defs/VersionSpec"
        }
      },
      "additionalProperties": false
    }
  ],
  "required": ["std.info"]
}
```

Annex B (informative)

Structured Parameters JSON Schema [strctparamjschm]

B.1 General [strctparamjschm.general]

1 This Annex defines the structured parameters capability schema (6.5) in terms of a *JSON Schema* (1.).

2 This JSON Schema can be referenced as the `$schema` field with URI value of `"std_param-1.0.0.json"`.

B.2 JSON Schema Specification [strctparamjschm.spec]

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "std_param-1.0.0.json",
  "title": "Structured Parameters Version 1.0.0 JSON Schema",
  "type": "object",
  "properties": {
    "$schema": {
      "description": "JSON Schema URI for the version of the structured parameters format.",
      "type": "string",
      "format": "uri"
    },
    "version": {
      "description": "The Structured Parameters format version.",
      "type": "string",
      "$ref": "#/$defs/Version"
    }
  },
  "patternProperties": {
    "arguments": {
      "description": "Application direct arguments.",
      "type": "array",
      "items": { "type": "string" }
    },
    "options": {
      "type": "object",
      "patternProperties": {
        "param": { "$ref": "#/$defs/Std.Param" },
        "source": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.Source"
        },
        "output": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.Output"
        },
        "include_dirs": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.IncludeDirs"
        },
        "library_dirs": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.LibraryDirs"
        },
        "define": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.Define"
        },
        "undef": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.Undef"
        },
        "language": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.Language"
        },
        "optimization": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.Optimization"
        },
        "vendor": {
          "$ref": "std_opt_core-1.0.0.json#/$defs/Std.Vendor"
        }
      }
    }
  },
  "additionalProperties": false,
  "oneOf": [{ "required": ["arguments"] }, { "required": ["options"] }],
  "$defs": {
    "Version": {
      "type": "string",
      "pattern": "^[0-9]+([.][0-9]+){0,2}$"
    },
    "Name": {
      "type": "string",
      "pattern": "^([a-z0-9_-]+[.])*([a-z0-9_-]+)$"
    },
    "StringOrArray": {
      "type": ["string", "array"],
      "items": { "type": "string" }
    },
    "Std.Param": {
      "description": "Recursive reference to one or more structured parameters files.",
      "type": "object",
      "properties": {
        "pre": { "$ref": "#/$defs/StringOrArray" },
        "post": { "$ref": "#/$defs/StringOrArray" }
      },
      "additionalProperties": false
    }
  }
}
```

Annex C (informative)

Structured Core Options JSON Schema [strctoptcorejschm]

C.1 General [strctoptcorejschm.general]

1 This Annex defines the structured parameter core options schema (Clause 7) in terms os a *JSON Schema* (1.).

2 This JSON Schema can be referenced as the `$schema` field with URI value of `"std_opt_core-1.0.0.json"`.

C.2 JSON Schema Specification [strctoptcorejschm.spec]

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "std_opt_core-1.0.0",
  "title": "Structured Core Options Version 1.0.0 JSON Schema",
  "patternProperties": {
    "source": { "$ref": "#/$defs/Std.Source" },
    "output": { "$ref": "#/$defs/Std.Output" },
    "include_dirs": { "$ref": "#/$defs/Std.IncludeDirs" },
    "library_dirs": { "$ref": "#/$defs/Std.LibraryDirs" },
    "define": { "$ref": "#/$defs/Std.Define" },
    "undef": { "$ref": "#/$defs/Std.Undef" },
    "optimization": { "$ref": "#/$defs/Std.Optimization" },
    "vendor": { "$ref": "#/$defs/Std.Vendor" },
    "language": { "$ref": "#/$defs/Std.Language" },
    "kind": { "$ref": "#/$defs/Std.Kind" }
  },
  "$defs": {
    "Std.Source": {
      "description": "The source files to process.",
      "type": "array",
      "items": {
        "description": "A file to process.",
        "type": "object",
        "patternProperties": {
          "name": { "type": "string" },
          "language": { "$ref": "#/$defs/Std.Language" },
          "kind": { "$ref": "#/$defs/Std.Kind" },
          "vendor": { "$ref": "#/$defs/Std.Vendor" }
        },
        "required": ["name"],
        "additionalProperties": false
      }
    },
    "Std.Output": {
      "description": "The output files to generate.",
      "type": "array",
      "items": {
        "description": "An output file.",
        "type": "object",
        "patternProperties": {
          "name": { "type": "string" },
          "kind": { "$ref": "#/$defs/Std.Kind" },
          "vendor": { "$ref": "#/$defs/Std.Vendor" }
        },
        "required": ["name"],
        "additionalProperties": false
      }
    },
    "Std.IncludeDirs": {
      "description": "Include directories.",
      "type": "array",
      "items": { "type": "string" }
    },
    "Std.LibraryDirs": {
      "description": "Library directories.",
      "type": "array",
      "items": { "type": "string" }
    },
    "Std.Define": {
      "description": "Define preprocessor symbols.",
      "type": "array",
      "items": {
        "type": "object",
        "patternProperties": {
          "name": { "type": "string" },
          "value": {
            "type": ["number", "integer", "string", "boolean", "null"]
          }
        },
        "required": ["name"],
        "additionalProperties": false
      }
    },
    "Std.Undef": {
      "description": "Undefine preprocessor symbols.",
      "type": "array",
      "items": { "type": "string" }
    },
    "Std.Language": {
      "description": "The language to interpret sources for.",
      "type": "object",
      "patternProperties": {
        "name": {
          "type": "string"
        }
      },
      "required": ["name"]
    },
    "Std.Optimization": {
      "description": "The optimizations to apply to different stages of the processing.",
      "type": "object",
      "patternProperties": {
        "compile": {
          "type": "string",
          "enum": ["off", "minimal", "speed", "space", "debug"]
        },
        "link": {
          "type": "boolean"
        },
        "vendor": { "$ref": "#/$defs/Std.Vendor" }
      }
    },
    "Std.Vendor": {
      "description": "Vendor defined specifications.",
      "type": "object",
      "patterProperties": {
        "^[a-z]+$": {
          "type": "object"
        }
      }
    },
    "Std.Kind": {
      "description": "The kind of data a source of output is.",
      "type": "string",
      "enum": ["text", "exec", "object", "dynamic_lib", "archive_lib"]
    }
  }
}
```

Bibliography

1. Internet Engineering Task Force. IETF RFC draft *JSON Schema: A Media Type for Describing JSON Documents*. Available from: `https://json-schema.org/draft/2020-12/json-schema-core.html`

Cross references

Each clause and subclause label is listed below along with the corresponding clause or subclause number and page number, in alphabetical order by label.

Index

Constructions whose name appears in *`monospaced italics`* are for exposition only.