Chapter

16 *CPP2SDL Migration Guide*

This chapter is a practical guide on how to convert an SDL specification that uses the H2SDL tool, which is not supported any longer, into an equivalent specification that uses the new CPP2SDL tool.

A complete description of the CPP2SDL tool can be found in "The CPP2SDL Tool" on page 757 in chapter 15, *The CPP2SDL Tool* .

Introduction

The purpose of this chapter is to help H2SDL users to smoothly migrate from the obsolete H2SDL tool to CPP2SDL. In addition, Telelogic Customer Support will be happy to assist in solving any migration problems not covered by this chapter.

The H2SDL tool provided automated support for accessing C code from within the Telelogic Tau SDL Suite. The CPP2SDL tool works according to the same fundamental principle, i.e. by automatically translating C/C++ declarations into SDL representations.

However, due to the difference between the C and C++ languages, CPP2SDL is not entirely backward compatible with H2SDL. This means that existing SDL systems that use H2SDL, may need to be updated when upgrading to CPP2SDL.

Reasons to Migrate

Apart from H2SDL being obsolete, there are several reasons to migrate from H2SDL to CPP2SDL:

- CPP2SDL has superior language support. Not only does it support C++, but it also covers a larger set of the C language.
- CPP2SDL allows the user to configure the translation from C/C++ to SDL by means of import specifications. This is a technique which often cuts the build time significantly, when large APIs are interfaced from SDL.
- With H2SDL, the user often has to edit input header files, to make them suitable for translation. This need is significantly reduced with CPP2SDL, since the import specification mechanism lets the user configure the translation of each C/C++ declaration individually.
- Translation rules implemented by CPP2SDL are considerably improved, compared to the ones implemented by H2SDL. These improvements are due to the implementation of language extensions oriented towards SDL2000.
- CPP2SDL has an improved reference generator, allowing more precise analysis.
- The SDL package generated by H2SDL may only be used at system level, while the SDL declarations generated by CPP2SDL may be injected at any level in the SDL hierarchy. This is possible by means of a new SDL-PR symbol in the SDL Editor.
- CPP2SDL supports a wider range of preprocessors and compilers to be used for preprocessing input header files.
- CPP2SDL is built with state-of-the-art compiler technology on top of a commercial C/C++ parser. This provides for rapid incorporation of future extensions to C/C++.

Migration Guidelines

This section describes the procedure of upgrading an SDL system from H2SDL to CPP2SDL usage. Required changes are grouped into subsections corresponding to the main reasons for the changes.

Note:

The CPP2SDL tool takes advantage of SDL2000 being case-sensitive, therefore you should make sure that your SDL system is possible to analyze in case-sensitive mode prior to starting the migration work.

The Telelogic Tau SDL Suite includes a tool that helps you convert your SDL system into a case-sensitive SDL system (see Update to case-sensitive SDL.

Update to case-sensitive SDL

The following steps will help you convert your SDL system into case-sensitive mode.

1. Make sure you use the standard Text Editor for editing text files in the SDL Suite.
2. Files should be write enabled. If not, the files will be updated but stored with the extension `.keep`.
3. Check the correctness of the system with the batch commands described in "Checking diagrams for duplicated object IDs" on page 206 in chapter 2, *The Organizer*.

   You should now be ready to convert to case-sensitive mode. Start with creating an extended cross reference file and a file with references to all keywords in the system.
4. Issue the following command:

   ```
   sdtbatch[.bat] -a systemPath/systemname.sdt -options
   optionFile.txt
   ```

   The file `optionFile.txt` should include the following lines of options to the Analyzer:

   ```
   [ANALYSEROPTIONS]
   SDLKeywordFile=True
   SetPredefinedXRef=True
   XRef=True
   CaseSensitiveSDL=False
   ```

   Two files will be created in the target directory of the system. They are called `casesensitive.xrf` and `casesensitive.key` and should be used in the next step.

5. Now you can start the update of the system by invoking the following command.

   ```
   sdtbatch[.bat] -changecase
   targetDir/casesensitive.xrf
   ```

   The result of the operation is sent to `sdtout` and contains information about what have been done, problems that have arisen and finally which files that was updated and saved. Since it is a lot of information you should probably pipe the output to a file.

   When the case is updated we recommend that you analyze the system again. Proceed as follow:

6. Change the last row in the `optionFile.txt` from above. The last row should now be:

   ```
   CaseSensitiveSDL=True
   ```

7. If your system uses the ctypes package and H2SDL you will need to use a special ctypes package that makes it possible to analyze in case-sensitive mode. Open the system in the Organizer and connect ctypes package to the file `<installation directory>/sdt/include/ADT/ctypes_migration.sun`.

8. Analyze the system by issuing the following command:

   ```
   sdtbatch[.bat] -a systemPath/systemname.sdt -options
   optionFile.txt
   ```

   There are known restrictions that may make it necessary to manually make further changes of the case in some places in the system. Known problems are:

   – Text in class symbols are ignored.

   – Words spanning two symbols using the possibility to divide a word with '_' are not updated.

   – When generating SDL PR files from SDL GR files some words are trimmed for spaces. Those may not be handled correctly.

We recommend that you update the system further until you have no known errors in the system when running in case-sensitive mode before continuing the process to migrate to CPP2SDL.

Changed Tool Integration

CPP2SDL is integrated with the Organizer and the SDL Editor in a slightly different way than H2SDL is. Instead of placing the header files one by one at root level in the Organizer view, it is now possible to define a set of header files which will be translated by CPP2SDL as a group. Each such set is organized by an import specification which controls what declarations should be translated and how they should be translated. The SDL/PR file that results from the processing of an import specification is represented by means of a PR symbol in the SDL Editor.

1. Remove all header files from the root level in the Organizer view. Also remove the symbol for the ctypes package.
2. In the extended heading of the SDL system diagram, remove the use clauses of the packages, which were generated by H2SDL for the header files. Also remove the use clause of the ctypes package.

> **Note:**
>
> The ctypes package must not be used with CPP2SDL as it is designed exclusively for H2SDL. A set of SDL/PR files plays the same role as the ctypes package and will be included by selecting an option in an import specification as described in step 5.

Now all dependencies on H2SDL have been removed, and the specification may be set up for using CPP2SDL instead.

3. Add an import specification at system level in the Organizer view. Do this by placing a PR symbol in the SDL system diagram. Then double-click the PR symbol, and select it to be a C import specification in the dialog that appears.
4. Add the removed header files to the import specification, by selecting the import specification and using the Add Existing command in the Organizer. The symbols for the added header files will appear below the import specification symbol in the Organizer.
5. Go to the CPP2SDL Options dialog for the import specification and set the option[1] to generate SDL representations for fundamental types.

When the above steps have been performed, the SDL specification should be equivalent to the original specification from a tool integration

point-of-view. However, while doing these changes there are a few optimizations that are enabled by CPP2SDL which should be considered. We will look at these later in "Configuring CPP2SDL Translation" on page 874.

Differences in Translation Rules

Although the translation rules implemented by CPP2SDL in many ways are similar to the ones implemented by H2SDL, they are not identical. Therefore, it may be necessary to do some modifications in the SDL specification.

The following sections follow the presentation in chapter 15, *The CPP2SDL Tool*.

Names

The names of generated SDL identifiers may differ in prefixes and suffixes. All references to such identifiers from the SDL specification should be updated, if needed.

1. Remove suffixes generated by H2SDL to handle case-sensitivity. These suffixes have the format "_*i*", where *i* is a natural number.
2. Remove prefixes generated by H2SDL to handle combinations of underscores that previously were not supported. These prefixes have the format "zz_UScr_*ij*_", where *i* and *j* are natural numbers. Also change the name of these identifiers to be the same as the corresponding C identifiers.
3. Change prefixes generated by H2SDL to handle SDL keywords. These prefixes have the format "zz_CCod_" and should be replaced with "keyword_".

Hint:

An alternative to changing all keyword prefixes manually, is to set the keyword prefix to be "zz_CCod_" in the CPP2SDL Options dialog.

1. This option plays the same role as the ctypes package did for H2SDL. When it is set, the generated SDL will include suitable SDL/PR files with SDL representations of fundamental C/C++ types and type declarators (e.g. int, char, pointers and arrays). If set at more than one specification, errors will occur. Therefore, set this option at the broadest scope in which it will be used.

Fundamental Types

The ctypes package used by H2SDL, is replaced by one of two sets of SDL/PR files, depending on the used input language; BasicCTypes and CPointer, or BasicC++Types and C++Pointer. They are included automatically in the SDL generated by CPP2SDL if the *Generate SDL representations for fundamental types* is set. Refer to "Fundamental Types" on page 779 in chapter 15, *The CPP2SDL Tool* for more information.

The table below describes the differences between the H2SDL ctypes package and the corresponding CPP2SDL SDL/PR files. The most obvious difference is that the SDL names of the sorts that represent fundamental C/C++ types are more consistent and intuitive when CPP2SDL is used.

Also note that the `CharStar` and `VoidStarStar` sorts of ctypes (representing `char*` and `void**` in C/C++) are not predefined in the CPP2SDL included SDL/PR files. See "Predefined Pointer Types" on page 871 and "Predefined Operators" on page 871.

C/C++ Fundamental Type	SDL Sort when using H2SDL	SDL Sort when using CPP2SDL
`signed int` `int`	`Integer`	`int`
`unsigned int` `unsigned`	`UnsignedInt`	`unsigned_int`

`signed long int`
`signed long`
`long int`
`long`
	`LongInt`	`long_int`

`unsigned long int` `unsigned long`	`UnsignedLongInt`	`unsigned_long_int`

`signed short int`
`signed short`
`short int`
`short`
	`ShortInt`	`short_int`

`unsigned short int` `unsigned short`	`UnsignedShortInt`	`unsigned_short_int`

C/C++ Fundamental Type	SDL Sort when using H2SDL	SDL Sort when using CPP2SDL
signed long long int signed long long long long int long long	LongLongInt	long_long_int
unsigned long long int unsigned long long	UnsignedLongLongInt	unsigned_long_long_int
char	Character	char
signed char	Character	signed_char
unsigned char	Octet	unsigned_char
char *	CharStar	N/A
float	Float	float
double long double	Real	double
bool	N/A	bool
wchar_t	N/A	wchar_t
void *	VoidStar	ptr_void
void **	VoidStarStar	N/A

4. Update all references to sorts corresponding to fundamental types, with the names of the corresponding SDL sorts generated by CPP2SDL.

> **Hint:**
>
> An alternative to changing all these references manually is to define a set of syntypes mapping the old names to the new ones.

Type Declarators

The names of SDL sorts representing pointer and array types have been changed to be more intuitive and shorter when CPP2SDL is used.

5. Update all references to sorts corresponding to pointer types by changing the old pointer prefix “Ref_*filename*_” to “ptr_”.

6. Update all references to sorts corresponding to array types by changing the old array prefix "CA_*filename_i*_X_" to "arr_*i*_", where *i* is the number of elements in the array.

As the H2SDL array prefix "CA_*filename_i*_X_" would result in very long SDL sort names when translating multi-dimensional arrays, H2SDL generates syntypes to shorten the sort names. These syntypes are named "MA_*filename_j*", where *j* is a natural number. Example 140 gives an example of this technique and the corresponding CPP2SDL translation.

Example 140: H2SDL translation of multi-dimensional array compared to CPP2SDL translation

C header:

```
int func(int[2][3][4]);
```

H2SDL translation:

```
newtype CA_filename_4_X_Integer /*#SYNT*/
  CArray(4,Integer);
endnewtype CA_filename_4_X_Integer;

newtype CA_filename_3_X_CA_filename_4_X_Integer
/*#SYNT*/
  CArray(3,CA_filename_4_X_Integer);
endnewtype CA_filename_3_X_CA_filename_4_X_Integer;

syntype MA_filename_0 /*#SYNT*/ =
 CA_filename_3_X_CA_filename_4_X_Integer
endsyntype;

newtype CA_filename_2_X_MA_filename_0 /*#SYNT*/
  CArray(2,MA_filename_0);
endnewtype CA_filename_2_X_MA_filename_0;

procedure func;
  fpar
    fpar_0 CA_filename_2_X_MA_filename_0;
  returns Integer;
external;
```

CPP2SDL translation:

```
NEWTYPE global_namespace /*#NOTYPE*/
  OPERATORS
    func : arr_2_arr_3_arr_4_int -> int;
/*#ADT(A(S) E(S) K(H))*/
ENDNEWTYPE global_namespace;EXTERNAL 'C++';
NEWTYPE arr_4_int CArray( 4, int);
/*#ADT(A(S) E(S) K(H))*/
```

```
ENDNEWTYPE arr_4_int;EXTERNAL 'C++';
NEWTYPE arr_3_arr_4_int CArray( 3, arr_4_int);
/*#ADT(A(S) E(S) K(H))*/
ENDNEWTYPE arr_3_arr_4_int;EXTERNAL 'C++';
NEWTYPE arr_2_arr_3_arr_4_int CArray( 2,
arr_3_arr_4_int);
/*#ADT(A(S) E(S) K(H))*/
ENDNEWTYPE arr_2_arr_3_arr_4_int;EXTERNAL 'C++';
```

7. If syntypes for multi-dimensional arrays are present ("MA_*filename_j*"), replace them with one "arr_*i*_" prefix for each array dimension.

Predefined Pointer Types

The H2SDL ctypes package includes the definition of the SDL sorts `CharStar` and `VoidStarStar` corresponding to the C/C++ pointer types `char*` and `void**`. These predefined sorts are not supported by CPP2SDL. Instead CPP2SDL will translate these pointer types using the `Ref` generator as they are needed.

8. Update all references to the obsolete sort `CharStar` to the translated sort `ptr_char`.

9. Update all references to the obsolete sort `VoidStarStar` to the translated sort `ptr_ptr_void`.

Predefined Operators

The ctypes package also defines a number of obsolete conversion operators for the `Ref` generator and the `CharStar` and `VoidStarStar` sorts, these are replaced with the `cast` operator according to the table below.

H2SDL operator	CPP2SDL operator	Comment
`ref2vstar`	`cast`	
`vstar2ref`	`cast`	
`ref2vstarstar`	`N/A`	Use `cast(cast( ))` instead, i.e `<type>*` to `void*` to `void**`
`cstar2cstring`	`cast`	Defined in CharConvert.pr
`cstring2cstar`	`cast`	Defined in CharConvert.pr

H2SDL operator	CPP2SDL operator	Comment
`cstar2vstar`	`cast`	
`vstar2cstar`	`cast`	
`cstar2vstarstar`	`N/A`	Use `cast(cast( ))` instead, i.e `<type>*` to `void*` to `void**`
`vstarstar2vstar`	`cast`	
`vstar2vstarstar`	`cast`	

10. If the `cstar2cstring` or `cstring2cstar` operators are used, add a PR symbol and connect it to the file <installation directory>/include/ADT/CharConvert.pr.

11. Replace all occurrences of the obsolete conversions operators with the `cast` operator according to the table above.

Enumerated Types

H2SDL has two alternative translations of enum declarations; using integer synonyms or using newtype literals. CPP2SDL only supports the latter of these alternatives.

If H2SDL was configured to translate enum literals to integer synonyms instead of newtype literals, and such a generated SDL synonym is used as an arithmetic expression, then the `EnumToInt` operator generated by CPP2SDL should be invoked on the newtype literal that corresponds to that synonym. See "Enumerated Types" on page 783 in chapter 15, *The CPP2SDL Tool* for more information.

Another difference in the translation of enumerated types is that H2SDL adds an `Enum_` prefix to the generated newtype, while CPP2SDL uses the name as it is.

12. Remove all `Enum_` prefixes from identifiers in references to newtypes representing enumerated types.

Functions

While H2SDL translates function prototypes to SDL procedures, CPP2SDL translates them to operators.

13. Convert all calls to procedures that represent C functions into operator calls by removing the `call` keyword.

14. If the Procedure call symbol is used to call procedures representing C functions, replace the Procedure call symbol with a Task symbol.

Note:

A procedure without parameters is called differently than an operator without parameters:

```
/* Calling a procedure p without parameters. */
task call p();
/* Calling an operator o without parameters. */
task o;
```

Variables

H2SDL does not translate C variables directly, but generates two procedures for getting and setting the value of the variable. CPP2SDL, on the other hand, generates external SDL variables, which is a tool-specific SDL extension.

15. For each accessed variable `v`, replace all procedure calls to `Set_v(`*`value`*`)` with the assignment `v :=` *`value`*. Replace procedure calls to `Get_v()` with the variable name `v`.

Note:

CPP2SDL will only translate variables when the Import Specification is placed in a SDL diagram that allows declaration of variables.

Structs and Unions

With CPP2SDL, the names of newtypes generated from structs and unions are retained, while H2SDL adds a `Struct_` or a `Union_` prefix to these names.

16. Remove all `Struct_` and `Union_` prefixes from identifiers in references to newtypes representing structs or unions.

Dynamic Memory Management

The definition of the Ref generator that was used with H2SDL only contained an operator (`Make!`) or literal (`Alloc`) for instance allocation,

while instance deallocation was made with a `Free` procedure, defined outside the generator. With CPP2SDL, the `Free` procedure has been replaced with a `free` operator in the Ref generator.

17. Convert all calls to the `Free` procedure into calls to the `free` operator by removing the **call** keyword and changing `Free` to `free`.

Configuring CPP2SDL Translation

While H2SDL performs a full translation of all supported C constructs to the system scope of the SDL specification, the Import Specification makes it possible to define what CPP2SDL translates and where to import it.

Additionally, it is also possible to use multiple Import Specifications to import C declarations to different scopes in the SDL specification.

> **Note:**
>
> The SDL language does not contain any construct like the C `#ifdef` construct, so take care not to import any declarations more than once if you are planning to use multiple Import Specifications.

Where to place the Import Specifications

For example, instead of always placing an import specification at system level, it should be placed at the most narrow scope that encloses all SDL usages of the declarations in the header files that are present under that import specification.

CPP2SDL Options

H2SDL used a centralized setting controlling which preprocessor and what preprocessor options to use for preprocessing input headers. CPP2SDL is more flexible, and makes it possible to specify these settings for each import specification.

1. Open the CPP2SDL Options dialog for each import specification, and enter the preprocessor and preprocessor options to use for preprocessing the headers grouped by that import specification.

Also note that H2SDL defines the macro `__H2SDL__` while preprocessing the C header files. When using CPP2SDL this macro is replaced by the `__CPP2SDL__` macro.

2. If the __H2SDL__ macro is used in your C headers change this to __CPP2SDL__.

> **Hint:**
>
> An alternative to changing the __H2SDL__ macro is to use the pre-processor options in the CPP2SDL Options dialog to define the __H2SDL__ macro.

For more information on the options available in the CPP2SDL Options dialog please refer to "Setting CPP2SDL Options in the Organizer" on page 762 in chapter 15, *The CPP2SDL Tool*.

What to Translate

The TRANSLATE section of the Import Specification controls which C identifier that should be translated, by default CPP2SDL will also translate any declarations that these identifiers are depending on.

With the TRANSLATE section it is also possible to add type declarators to types and supply prototypes for ellipsis functions making these available in SDL.

Read more about the TRANSLATE section in "Import Specifications" on page 771 in chapter 15, *The CPP2SDL Tool*.

New Build Options

Contrary to H2SDL, CPP2SDL requires the generated SDL to be analyzed as case sensitive SDL.

3. In the Analyzer Options dialog, set the Case sensitive SDL option.

It should now be possible to analyze the SDL specification.

When the SDL specification has been accepted by the Analyzer, it should also be possible to generate code without further modification. Note, however, that using the Targeting Expert is the recommended way of generating code when CPP2SDL is used. This means that if a makefile or makefile template was used with H2SDL to link the external object files to the generated application, this could now be achieved in an easier way using the Targeting Expert. For more information, see "Generated Makefile" on page 2922 in chapter 60, *The Targeting Expert*.

Chapter 16 CPP2SDL Migration Guide