FAQ 746

How to Configure a Reusable Targetlink Subsystem and Its Identifiers for Use in Multiple CGUs

Question

I want to use reused functions in multiple TargetLink subsystems or multiple subsystems for incremental code generation. How do I have to configure a reusable function and the identifiers of instance structure variables as well as the identifiers of the types of instance structure variables? How can I avoid linker errors when sharing reused functions between multiple code generation units (CGUs)?

Solution

If you share reused functions between multiple CGUs, you have to ensure the following:

(1) In the Function block of the reusable subsystem, specify a fixed subsystem ID and a DD Module object. Potential problems regarding a missing subsystem ID are indicated by warning W30034.

(2) In the Data Dictionary, specify a DD ModuleOwnerShip object to configure which CGU is used to generate the production code of the reusable subsystem.

Note: If you have several subsystems whose code you want to generate into the same module but no CGU contains all the desired subsystems, refer to FAQ 748.

(3) Instance structure variables created for different CGUs must have different identifiers to ensure instance-specific behavior.

(4) The struct types of the instance structure variables and so the type of the formal parameter of the reused function the instance structure variables must have the same identifiers, because all CGUs use the same implementation of the reusable function.

Points 3 and 4 on naming are implemented with one DD StructTemplate object and one SubStructTemplate. If the templates have not been created yet, create a DD StructTemplate and a SubStructTemplate object in the Data Dictionary. To do this, use the context menu of the desired DD TemplateGroup object below /Pool/Templates.

Then make the following settings:

- StructTemplate
  (The DD StructTemplate object is named FcnReuseMainStruct in this example):
  - /Pool/Templates/Structs/FcnReuseMainStruct/Filter/StructSpec=FcnReuse
  - /Pool/Templates/Structs/FcnReuseMainStruct/Settings/NameTemplate=ISV_IR_$B_$N
  - /Pool/Templates/Structs/FcnReuseMainStruct/Settings/TypedefNameTemplate=ISV_$I_tp
- SubStruct template
  (The DD StructTemplate object is named FcnReuseSubStruct in this example):
  - /Pool/Templates/Structs/FcnReuseSubStruct/Filter/StructSpec=FcnReuse

- /Pool/Templates/Structs/FcnReuseSubStruct/Settings/NameTemplate= ISV_IR_$B_$N_$(Section)
- /Pool/Templates/Structs/FcnReuseSubStruct/Settings/TypedefNameTemplate= ISV_$I_$(Section)_tp

Explanations of the name macros used in this context:

- **$I** expands to the subsystem ID of the reused subsystem, but it is not CGU-specific (except if the reusable subsystem is specified as incremental CGU itself).

  Note: You could use $S, but dSPACE recommends $I.
  $S consists of the character S, the associated TargetLink subsystem ID ($I), and an integer number that identifies the subsystem. Usually, the integer is used to avoid ambiguous names, but in this context $I is already required to be unique (see (1) above). Therefore, $I is simpler in this use case.

- **$R** allows the Code Generator to change an identifier to make it unambiguous within one CGU if required, to avoid naming conflicts.
  This makes it easier to achieve a successful build without errors.

- **$B** expands to the name of the instance of the reused subsystem.

  This might be different if all instances of the reused system have different names. If the names are the same, $B evaluates to that name.
  Therefore, using $B for instances with different names is helpful if you want to identify which ISV struct variables belong to a specific instance. In this case, you do not have to use $R, which would lead to unambiguous names, but it would be difficult to associate the names with the corresponding instance of the reusable system.

- **$N** expands to the name of the CGU and therefore supports CGU-specific names.

- **$(Section)** expands to the number of the function reuse substructure. An instance structure can be divided into several substructures. Criteria include initialized/not initialized, const/not const, etc. For more information, refer to the TargetLink user documentation. $(Section) evaluates to the number of the section. The section numbers start with 0 and are incremented with an increment of 1.

Note: You do not have to use precisely the same name templates described above. However, all the name macros used in the name templates described above must be used to ensure proper functionality in all situations, regardless of the specific modeling (“reuse in reuse” or whether the instances have different names).

Related FAQs

- FAQ 774 – Module Ownership, Modules, Code Generation Units, Stub Code
- FAQ 748 – How to generate common *.h/.c files which are used in several CGUs?

FAQ Overview

http://www.dspace.com/go/faq

Support

To request support, please use the form at http://www.dspace.com/go/supportrequest

Updates and Patches

Software updates and patches are available at http://www.dspace.com/go/patches.
dSPACE strongly recommends to use the most recent patches for your dSPACE installation.

Important Notice

dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany