ETAS ASCET-SE V6.4
EHOOKS Target

User Guide

Copyright

ASCET-SE V6.4 EHOOKS Target – User Guide R08 EN – 02.2023

Contents

1   Safety and Privacy Information

In this chapter, you can find information about the intended use, the addressed target group, and information about safety and privacy related topics.

Please adhere to the ETAS Safety Advice (**Help > Safety Advice** in the ASCET Component Manager) and to the safety information given in the user documentation.

ETAS GmbH cannot be made liable for damage which is caused by incorrect use and not adhering to the safety messages.

1.1   Intended Use

The ASCET tools support model-based software development. In model-based development, you construct an executable specification -– the model -– of your system and establish its properties through simulation and testing in early stages of development. When the model behaves as required, it can be converted automatically to production quality code via ASCET-SE.

The EHOOKS Target of ASCET-SE is a tool for the following purposes:

- enable interaction between ASCET-SE and EHOOKS
    - map ASCET messages onto bypass hooks provided by EHOOKS
    - map ASCET processes onto bypass containers provided by EHOOKS
    - use services provided by external libraries and/or the ECU itself in ASCET-generated code
- on-target prototyping using the ECU as the prototyping platform

1.2   Target Group

You are a trained function developer who wants to do on-target prototyping using the ECU as the prototyping platform. You have knowledge of software development using ASCET and the use of the EHOOKS tools.

1.3   Classification of Safety Messages

The safety messages used here warn of dangers that can lead to personal injury or damage to property:

> ⚠️ **DANGER**
>
> **DANGER** indicates a hazardous situation that, if not avoided, will result in death or serious injury.

> ⚠️ **WARNING**
>
> **WARNING** indicates a hazardous situation that, if not avoided, could result in death or serious injury.

> ⚠️ **CAUTION**
>
> **CAUTION** indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.

> ***NOTICE***
>
> ***NOTICE*** indicates a situation that, if not avoided, could result in damage to property.

1.4 Safety Information

Please adhere to the ETAS Safety Advice and to the following safety information to avoid injury to yourself and others as well as damage to the device.

> ⚠️ **CAUTION**
>
> **Risk of harm or property damage**
>
> Wrong word size and/or compiler division lead to wrong compilable code. Wrong compilable code may lead to unpredictable behavior of a vehicle or test bench.
> When working with the EHOOKS target, you must create your bypass in a way that word size and compiler division match the selected EHOOKS-DEV back end to avoid wrong compilable code.

In addition, take all information on environmental conditions into consideration before setup and operation (see the documentation of your computer, hardware, etc.).

Further safety advice for this ETAS product is available in the following formats:

- in electronic form in the installation files:
  `Documentation\ETAS Safety Advice.pdf`

- Safety information given in the ASCET-SE User Guide and in the EHOOKS documentation

- ASCET V6.4 safety manual, available at ETAS upon request

1.5   Privacy Notice

Your privacy is important to ETAS so we have created the following Privacy Statement that informs you which data are processed in ASCET, which data categories ASCET uses, and which technical measure you have to take to ensure the users' privacy. Additionally, we provide further instructions where this product stores and where you can delete personal data.

1.5.1   Data Processing

Note that personal data or data categories are processed when using this product. As the controller, the purchaser undertakes to ensure the legal conformity of these processing activities in accordance with Art. 4 No. 7 of the General Data Protection Regulation (GDPR). As the manufacturer, ETAS GmbH is not liable for any mishandling of this data.

1.5.2   Data and Data Categories

Note that this product creates files containing file names and file paths, e.g. for purposes of error analysis, ensuring correct deinstallation, referencing source libraries, or for communicating with third party programs.

These file names and file paths may contain personal data, if they refer to the current user's personal directory or subdirectories (e.g., `C:\Users\<UserId>\Documents\...`).

If you do not want personal information to be included in the generated files, please make sure of the following:

- The workspace of the product points to a directory without personal reference.

- All settings in the product (see the menu option **Tools > Options** in the product) refer to directories and file names without personal reference.

- All project settings in the product (see the menu option **File > Properties** in the ASCET project editor) refer to directories and file names without personal reference.

- Windows environment variables (such as the temporary directory) refer to directories without personal reference because these environment variables are used by the product.

In this case, please also make sure that the users of this product have read and write access to the newly set directories.

When using the ETAS License Manager in combination with user-based licenses, particularly the following personal data or data categories are recorded for the purpose of license management:

- User data: UserID

- Communication data: IP address

As an option, the following personal data or data categories in particular may be recorded for the purpose of assisting development:

- Problem Report, see section 1.5.4 on this page.

When using the ASCET add-on ASCET -DIFF, particularly the following personal data or data categories are recorded for the purposes of user-specific settings and user-specific log files:

- User data: UserID

1.5.3   Technical and Organizational Measures

This product does not itself encrypt the personal data that it records. Ensure that the data recorded is secured by means of suitable technical or organizational measures in your IT system, e.g. by using classic anti-theft and access protection. Personal data in generated files can be deleted by tools in the operating system.

1.5.4   Description of Problem Report

Purpose: When an error occurs, ASCET offers to send an error report to ETAS for troubleshooting. ETAS uses the personal information to have a contact person in case of system errors.

Personal Data: The problem report may contain the following personal data or data category.

- user data
  * name and address entered during the installation process
  * UserID
- communication data
  * IP address

Additionally to the problem information that is entered by the users themselves, ASCET collects the available product-related log files in a ZIP archive to support the bug fixing process at ETAS. The ZIP file is named by using the pattern `EtasLogFiles<index number>.zip` and stored in the ETAS-specific log files directory.

This automatically created ZIP file contains the following:

- product-related log files created at installation time (necessary for uninstall action)

- ETAS log files stored in the ETAS log files directory matching the file name pattern `*.log` recursive registry export of ETAS (32bit)-key (and sub keys): `HKEY_CURRENT_USER\Software\ETAS`

- registry export of ETAS (32bit)-key (and sub keys): `HKEY_LOCAL_MACHINE\Software\ETAS`

All ETAS-related log files in the ETAS-specific log files directory and the ZIP archives created by the Problem Report feature can be removed after closing all ETAS applications if they are no longer needed.

2   About EHOOKS Target for ASCET-SE

Welcome to the EHOOKS Target for ASCET!

The EHOOKS Target allows you to use ASCET to build software for on-target bypass hooks and integrate it with existing ECU software using ETAS' EHOOKS tools.

This guide explains the following topics:

- how to install the EHOOKS Target (section 2.1 on this page)
- the basic concepts behind ASCET and EHOOKS interaction (section 2.2 on the next page)
- how to configure an ASCET project to use the EHOOKS Target (section 3.1 on page 17)
- how to map ASCET messages onto hooks, and processes into bypass containers provided by EHOOKS (section 3.2 on page 29 and section 3.3 on page 68)
- how to prepare calibration (chapter 4 on page 75)
- how to interact with EHOOKS control variables (chapter 5 on page 78)
- how to use services provided by external libraries and/or the ECU itself in ASCET-generated code (chapter 6 on page 81 and chapter 7 on page 100)
- how to update projects from old EHOOKS-DEV versions (chapter 8 on page 105)

2.1   Installation

This version of the EHOOKS Target requires the following products:

Product	Version
EHOOKS Dev Front End	V4.10 / V4.11 / V4.12 / V5.0
EHOOKS Dev Back End	V4.10 / V4.11 / V4.12 / V5.0
ASCET	V6.4.7

You must install the EHOOKS Front End and ECU Back End for each ECU you want to use for on-target prototyping.

2.1.1   Installation

The EHOOKS Target is part of ASCET-SE. When you install ASCET-SE, you must select the EHOOKS-enabled ECU target to install EHOOKS Target.

2.1.2   After Installation

Unlike other ASCET targets, the EHOOKS Target does not need to know which compiler and operating system are required. Compilation and OS integration issues for the target ECU are handled by EHOOKS.

Furthermore, ASCET does not need to be told where EHOOKS is installed on your system; the EHOOKS Target will find the EHOOKS tools automatically.

It is possible to have more than one EHOOKS Versions installed on the same host PC as ASCET-SE for the EHOOKS Target to work correctly. ASCET uses the version that is selected in the ASCET options window, "Targets\EHOOKS\Build" node (see Figure 3.4 on page 21), "External Build Tool" list.

2.2   Understanding ASCET/EHOOKS Integration

The ASCET EHOOKS Target provides a special ASCET-SE target that generates code for use as on-target bypass functions suitable for integration with an EHOOKS-prepared ECU. The EHOOKS Target can also transparently run the EHOOKS-DEV tool chain to integrate the generated code with ECU software with access to only the ECU hex and A2L files.

2.2.1   Typical Workflow

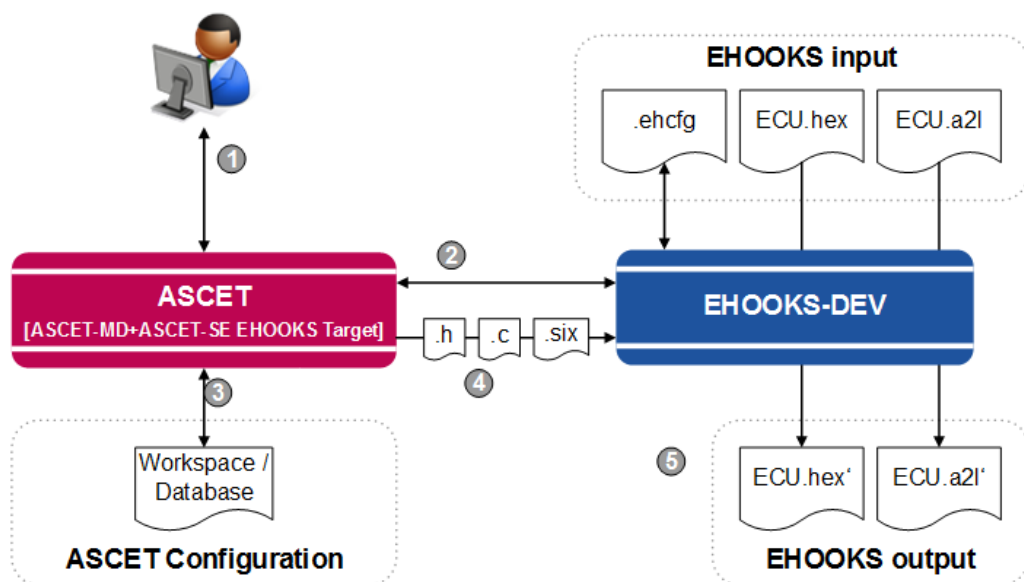Figure 2.1 shows the standard workflow when using the EHOOKS Target:

Figure 2.1: Workflow for ASCET/EHOOKS Development

A. You design ASCET models for your bypass functionality and integrate them into an ASCET project.

B. You configure the EHOOKS target for your ASCET project. ASCET will interact with EHOOKS to create a `*.ehcfg` configuration file.

C. ASCET stores the information about which parts of the ASCET model are hooked onto which ECU variables in the database or workspace.

D. ASCET generates code from the model as normal, but also the code and configuration files (SCOOP-IX) necessary to interface ASCET code with EHOOKS.

E. ASCET runs the EHOOKS build process to automatically generate new `.hex` and `.a2l` files that include your bypass functionality.

2.2.2 On-Target Bypass Concepts

On-target bypass allows run-time control of whether the original value calculated by the ECU or a value calculated by a bypass function running on the ECU is used for subsequent calculations as shown in Figure 2.2.
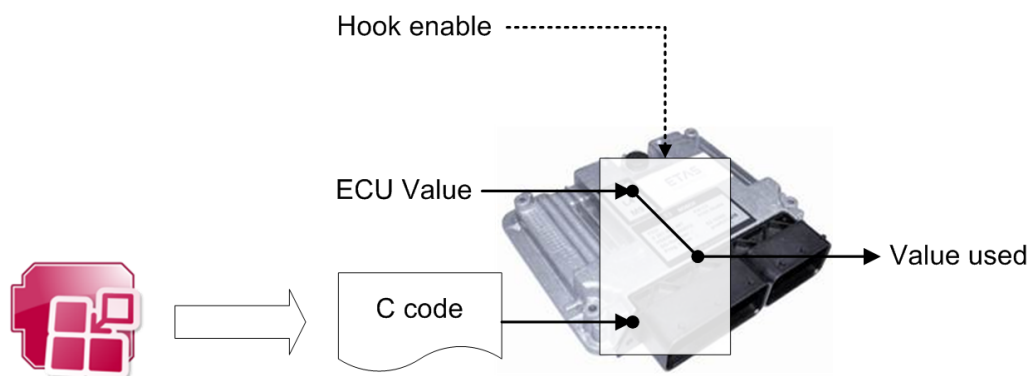
Figure 2.2: On-target bypass hooks with ASCET-generated C code

When the ECU is built, the ECU supplier decides which values can be switched between the ECU value and the bypass value, and creates a *hook* to allow the choice to be made. Hooks are therefore writeable values in the ECU software. The EHOOKS-PREP tool from ETAS allows ECU suppliers to choose and insert hooks into the ECU software; see the EHOOKS-DEV User Guide of your EHOOKS installation. The ECU is also prepared to include placeholders called *dispatch points*, into which bypass code can be placed. Dispatch points are typically found in existing ECU functions or OS tasks.

To use a hook you need to provide a bypass function and the associated EHOOKS configuration:

- which data is read

- which data is written

- when does the bypass function run

You could do this by hand, however, the EHOOKS Target allows functions to be developed as ASCET models.

2.2.3   ASCET Models as Bypass Functions

On-target bypass using the EHOOKS Target follows the same basic principles as bypass using ASCET-RP: the bypass model interacts with the ECU over the message interface. The ECU *sends* messages to the bypass function and *receives* messages that contain the bypass values from the bypass function, as shown in Figure 2.3 on this page.
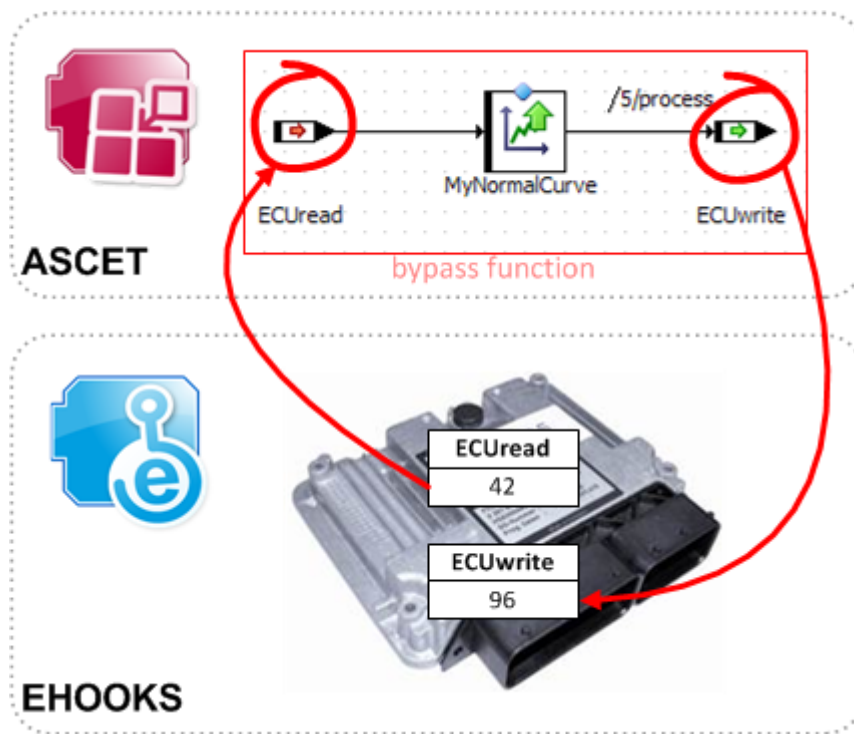
Figure 2.3: ECU sending and receiving messages from the bypass function

The EHOOKS Target interacts with the EHOOKS-DEV tool to allow the ASCET on-target bypass model to be configured to have access to one or more input variables from the ECU software (ECU measurements) and to write to one or more hooked ECU write variables. Each ASCET model can contain one or more bypass functions containing the processes of the ASCET model in which the hooked variables are read and written.

The EHOOKS Target can also use the features of EHOOKS-DEV to enable the introduction of new calibration parameters for the on-target bypass function. ASCET model elements that need to be calibrated must be assigned the scope "Exported" in the element's properties editor. This is because EHOOKS has to generate the data structures so it can integrate them with calibration parameters that already exist on the ECU. The EHOOKS Target tells EHOOKS-DEV what elements need to be generated for calibration using a SCOOP-IX (`*.six`) file.

2.2.4   Key Features of the EHOOKS Target

Key features of the EHOOKS Target are:

No special modeling required: It is not necessary to modify your models to work with the EHOOKS target. Models can be used unmodified with EHOOKS for complex internal bypass. It is only necessary to configure an EHOOKS target for a project and hook model messages onto hooks provided by the ECU.

No changes to generated code: ASCET generates identical C code from the model as it would when generating code for an embedded ECU. This means that the code for modules and classes is not modified in any way for the EHOOKS target. ASCET interfaces to EHOOKS by generating *bypass functions* that set up the context for ASCET on entry and tear down the context on exit.

No special memory configuration needed: EHOOKS does not have the notion of differing memory sections - there is simply code space, variable space and parameter space. ASCET ignores any memory sections definitions declared in model configuration.

No need to know the target ECU or compiler: ASCET does not need to know what micro-controller is being used in the ECU or what compiler needs to be used for building the bypass functions for integration - ASCET just sees a special *EHOOKS target*. Any EHOOKS-supported ECU can be used as an EHOOKS target. Details of supported ECUs can be obtained by contacting ETAS.

⚠️ **CAUTION**

Risk of harm or property damage

Wrong word size and/or compiler division lead to wrong compilable code. Wrong compilable code may lead to unpredictable behavior of a vehicle or test bench.
When working with the EHOOKS target, you must create your bypass in a way that word size and compiler division match the selected EHOOKS-DEV back end to avoid wrong compilable code.

Automatic conversion between ECU types and model types: ASCET automatically converts between ECU types and model types.

One-click ECU rebuild: ASCET generates code, the EHOOKS configuration, and runs the EHOOKS build process with a single mouse click.

2.2.5   Summary

ASCET automatically does the following:

- Add configuration information to the EHOOKS configuration file to integrate the bypass function, including telling EHOOKS-DEV what file to include in the ECU build.

- Create a SCOOP-IX file defining all global data, measurements and calibration parameters required for the bypass functions.

- Generate code that implements the ASCET model. The code is identical in structure and content to the code generated for all other ASCET-SE targets.

- Generate bypass functions that integrate ASCET-generated code with the EHOOKS-generated interface.

- Runs the EHOOKS-DEV tool to integrate the bypass functions with the ECU.

EHOOKS-DEV automatically does the following:

- Use the SCOOP-IX (`*.six`) file to generate ASCET-compatible data structures for all parameters.

- Use the EHOOKS (`*.ehcfg`) configuration file to generate an interface to the ECU data for the ASCET-generated bypass functions.

- Integrates all the source generated by both ASCET and EHOOKS-DEV with the existing ECU hex file.

2.3   Finding Out More

If not specified otherwise during installation, the following documentation is available after installing ASCET and ASCET-SE. Most PDF manuals are available in the `ETAS\ETASManuals\ASCET V6.4` folder, accessible via the Windows Start menu, **E > ETAS > Online manuals**.

- ASCET-SE user guide (`ASCET-SE V6.4 User Guide.pdf`)

- EHOOKS Add-On user guide (this manual; `ASCET-SE V6.4 EHOOKS Add On User Guide.pdf`)

- ASCET online help (accessible via the **Help** menu and <F1> in the ASCET windows)

- ASCET Getting Started (`ASCET V6.4 Getting Started.pdf`)

- ASCET Installation Guide (`ASCET V6.4 Installation.pdf`)

- ASCET Icon Reference Guide (`ASCET V6.4 Icon Reference Guide.pdf`)

- ASCET AUTOSAR User Guide (`ASCET V6.4 AUTOSAR User Guide.pdf`)

- AUTOSAR to ASCET Importer User Guide (`ASCET V6.4 AUTOSAR To ASCET Converter User Guide.pdf`)

> **ℹ Note**
>
> The cooperation of ASCET and AUTOSAR requires the installation of the ASCET-SE target `ANSI-C`.

When you install ASCET-RP, ASCET-SCM, or ASCET-DIFF, further documentation is available:

- ASCET-RP
  - user guide (`ASCET-RP V6.4 User Guide.pdf`)
  - separate online help; accessible via the **Help** menu and <F1> in the hardware configurator

- ASCET-SCM
  - online help (integrated in the main ASCET online help)

- ASCET-DIFF
  - ASCET-DIFF Getting Started; accessible via the Windows Start menu
  - separate online help; accessible via the **Help** menu and <F1> in the ASCET-DIFF windows

The EHOOKS-DEV User Guide (`EHOOKS-DEV V<x>.<y> User Guide.pdf`) and other EHOOKS documentation is available in the `Manuals` subfolder of your EHOOKS-DEV Front End and Back End installations.

3   Getting Started with an EHOOKS Project

You are now ready to create an ASCET project that uses an EHOOKS target.

Before you start, you must have the following *mandatory* items from your ECU supplier:

- the ECU `*.hex` file, pre-prepared for EHOOKS use, for the ECU you want to use for on-target prototyping

- the `*.a2l` file, pre-prepared for EHOOKS use, for the ECU you want to use for on-target prototyping

- the password for the `*.a2l` file (if is it password-protected)

Use of advanced capabilities of the EHOOKS Target, for example use of external or on-ECU services, requires some or all of the following *optional* items from your ECU supplier:

- a `services.ini` file from your ECU supplier defining the services available on the ECU

- a `*.ini` file from your ECU supplier defining the interpolation routines available for use on the ECU

- an ASCET workspace (or database) from your ECU supplier defining the model interface for library functions that are available for use on the ECU

- C source code files and/or pre-compiled libraries for your ECU that implement service routines

Further information about what is required and when can be found in chapters 6 and 7.

3.1   Project Administration

A new EHOOKS project is created with the following steps:

A. in ASCET: create an ASCET project for an EHOOKS target (section 3.1.1)

B. in ASCET: specify which EHOOKS `*.ehcfg` configuration file ASCET will use (section 3.1.2)

C. in ASCET: configure ASCET-EHOOKS interaction (section 3.1.3)

D. in EHOOKS: select the `*.hex` and `*.a2l` files EHOOKS will use (section 3.1.4)

The following sections explain these steps in more detail.

3.1.1   Creating an ASCET/EHOOKS Project

You must create an ASCET project in which to build your bypass functionality. You can use an existing project or create a new one.

To define an EHOOKS project

The project needs to be configured to target an EHOOKS prepared ECU as follows:

1. Create and open a project as described in the ASCET online help.

2. In the project editor, select **File > Properties** (or use <Ctrl> + <p>) to open the "Project Properties" window.

3. Go to the "Build" node and select EHOOKS as the target for the build as shown in Figure 3.1 on this page.

4. In the "Code Generator" combo box, select a code generator.

    Two code generators are available,
    `Object Based Controller Implementation` and
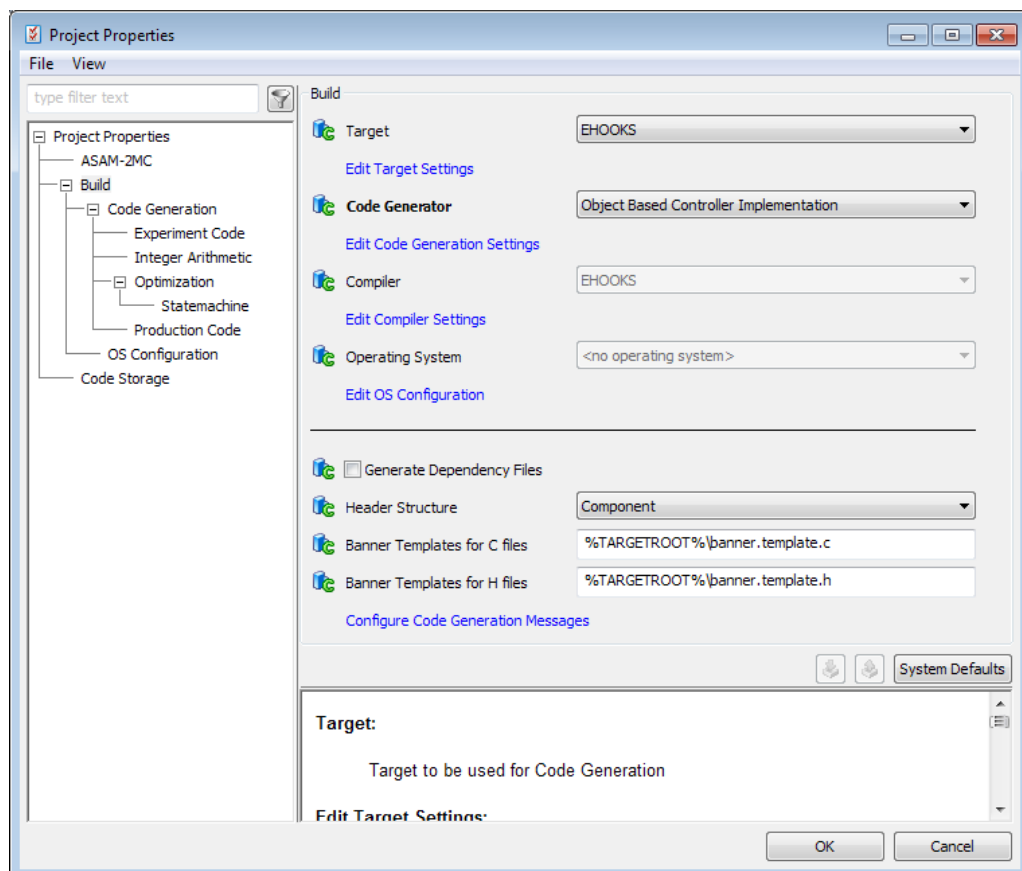   `Object Based Controller Physical`.

Figure 3.1: Configuring a project to use an EHOOKS target

When you set up the project to use the EHOOKS target, a new tab appears in the project editor (see Figure 3.2 on this page). This new tab replaces the "OS" tab; here you do all the configuration that is specific to EHOOKS projects in ASCET.
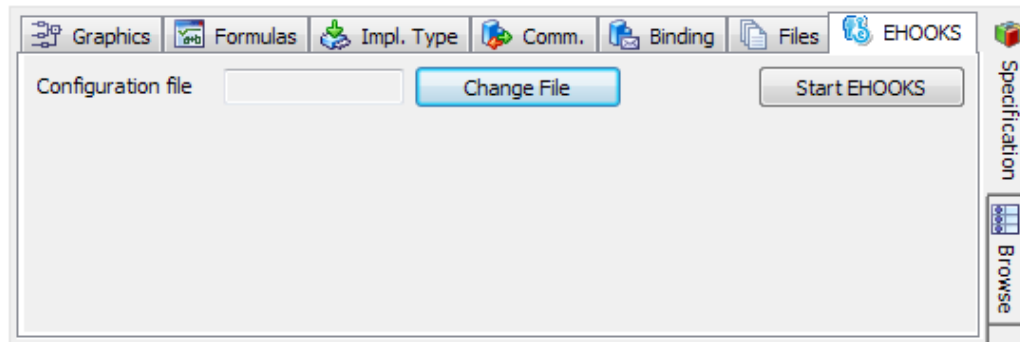

Figure 3.2: "EHOOKS" tab in the project editor (no configuration file selected)

3.1.2   Specifying the Configuration File Location

Each project that uses the EHOOKS target must be associated with an EHOOKS configuration file (`*.ehcfg`).

> **ℹ️ Note**
>
> You must associate your ASCET project with an EHOOKS configuration file before you can do any further configuration.

You can choose an existing EHOOKS configuration file or create a new one. If you use an existing EHOOKS configuration file, any pre-existing configuration items will be preserved. When ASCET generates EHOOKS configuration information in the file, only the parts owned by ASCET are modified. Non-ASCET-generated EHOOKS configuration is unchanged.

To select an EHOOKS configuration file

1. In the project editor, go to the "EHOOKS" tab.

2. In the "EHOOKS" tab, click the **Change File** button.

   The Windows file selection window opens. The file extension `*.ehcfg` is preselected.

3. Select your EHOOKS configuration file and click **Open**.

   Path and name of the EHOOKS configuration file are shown in the "Configuration file" field at the top of the "EHOOKS" tab. The sub-tabs "Scheduling", "Input" and "Output" appear. See also Figure 3.3 on the next page.
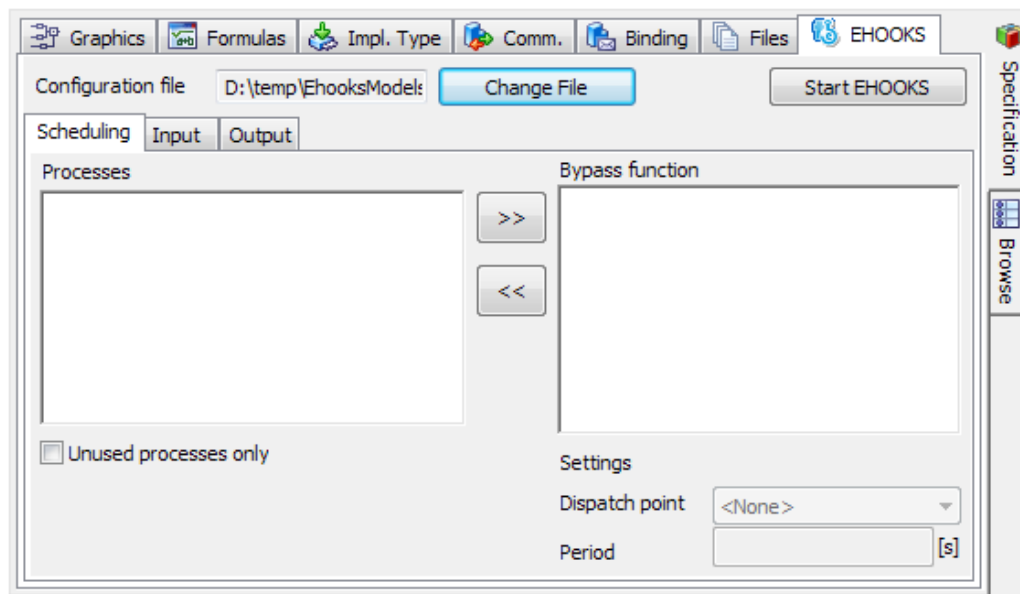
Figure 3.3: "EHOOKS" tab with EHOOKS configuration file

3.1.3   Configuring ASCET-EHOOKS Interaction Settings

When you have associated an EHOOKS configuration file with the ASCET project, you need to configure how ASCET interacts with EHOOKS.

3.1.3.1 EHOOKS Build Options

The EHOOKS Target uses the EHOOKS `toolchaindriver` program to re-build the ECU.

Any options that you want ASCET to pass to the `toolchaindriver` can be entered in the ASCET options window, "Targets\EHOOKS\Build" node (see Figure 3.4 on the next page), "Build Tool Options" field. The values are passed directly to the `toolchaindriver` without any modification and must be valid EHOOKS options.

Permitted options are listed in the EHOOKS-DEV user guide, section "EHOOKS-DEV Command Line Usage".
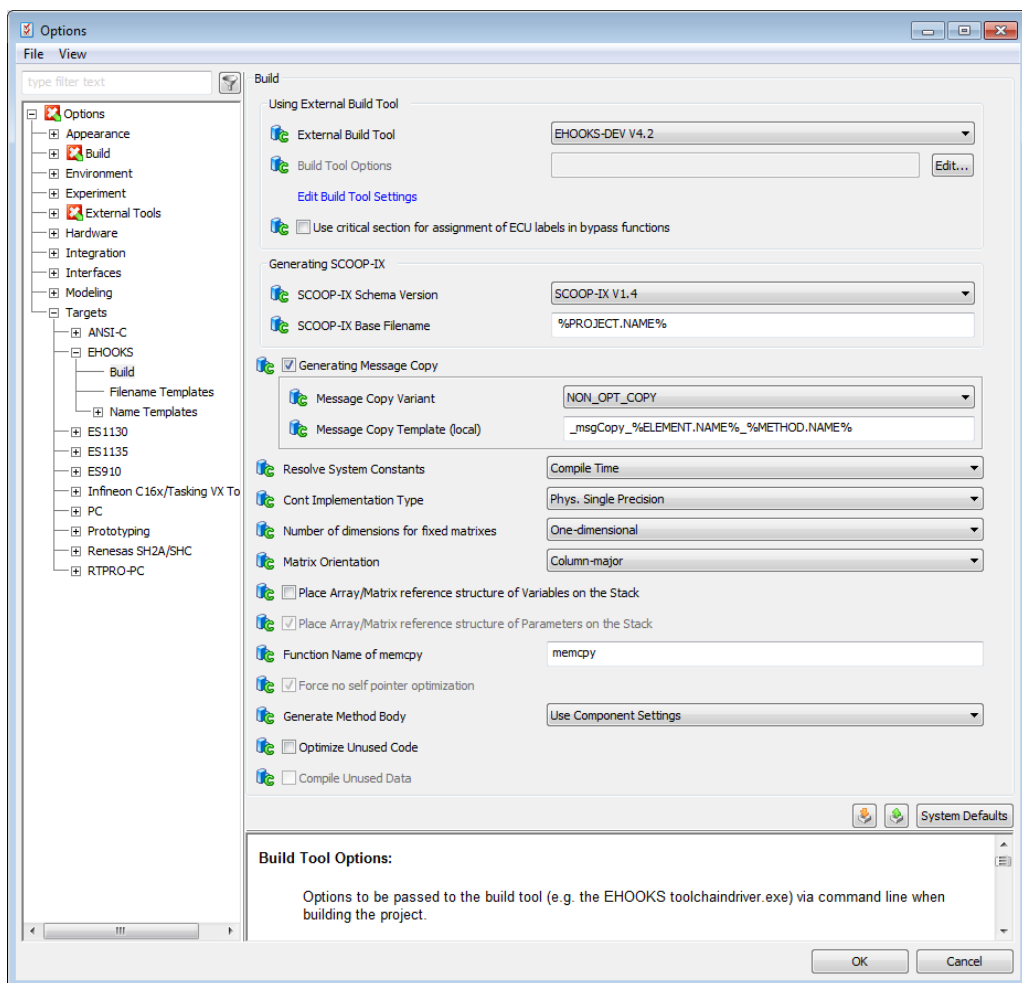
Figure 3.4: Build options for the EHOOKS target

3.1.3.2 Global Name Space Prefix

Global names generated by ASCET will not clash with names used by the ECU because EHOOKS works with a compiled HEX image.

However, the names that you use in your project may clash with the symbolic names of elements used on the ECU and stored in the `*.a2l` file.

To prevent this, ASCET allows the definition of a user-defined prefix that is, by default, added to all global data elements generated. The prefix is defined in the ASCET options window, "Targets\EHOOKS\Name Templates" node (see Figure 3.5 on this page); it is added to the default templates for element names and the element display names provided in the "Targets\EHOOKS\ Name Templates\ASAM-2MC" node (see Figure 3.6 on the next page).

By default, the module and class instance names are combined like in the following example with prefix **BP_**:

- global variable:

  BP_<*varname*>

- local variable in module:

  BP_<*varname*>.<*module_inst_name*>

- local variable in single-instance class in module:

  BP_<*varname*>.<*class_inst_name*>.<*module_inst_name*>
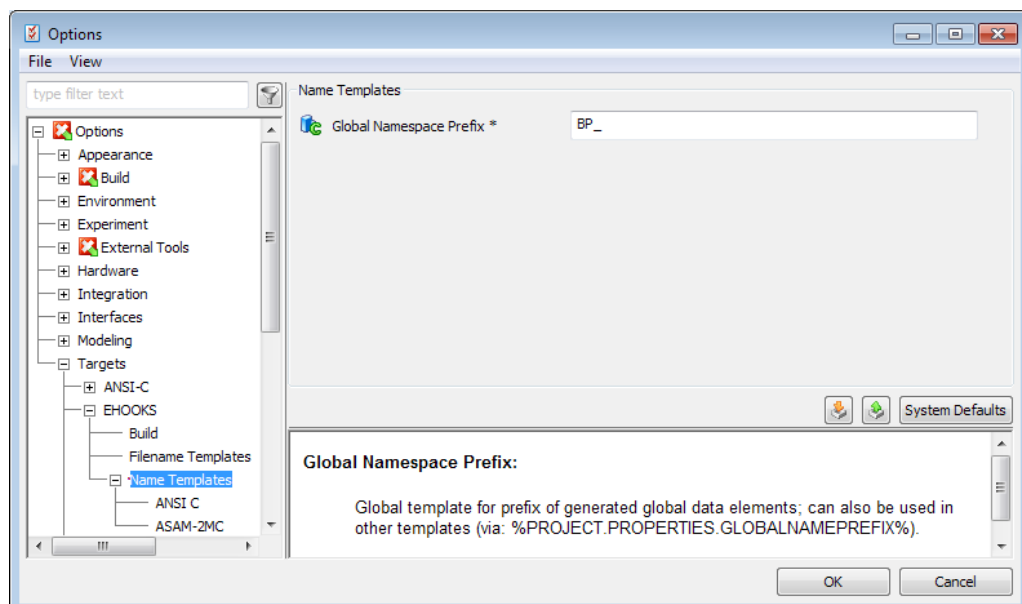
For parameters, names are generated the same way.

Figure 3.5: Name Templates options for the EHOOKS target

By default, the element name and the element display name in the `*.a2l` file are the same. However, you can change the default by editing the default

name templates (see also section 3.1.3.3 on this page). If you do so, make sure to avoid clashes with the symbolic names of elements used on the ECU and stored in the `*.a2l` file.

3.1.3.3 ASAM-MCD-2MC Names

In ASAM-MCD-2MC files, the labels of characteristics and measurements usually are dot-separated strings containing the names of the items on the instance path. There are two frequently used variants:

- element first, root node last

  This variant is suitable if the labels are very long and the display fields are left-justified and may be unable to show the complete labels.

- root node first, element last

  This variant is suitable to sort labels, or to structure them in a tree.

For local and exported elements, ASCET allows to configure the ASAM-MCD-2MC names, i.e. `IDENTIFIER` and `DISPLAY_IDENTIFIER`, in the "Targets\EHOOKS\Name Templates\ASAM-2MC" of the ASCET options window (see Figure 3.6 on this page).
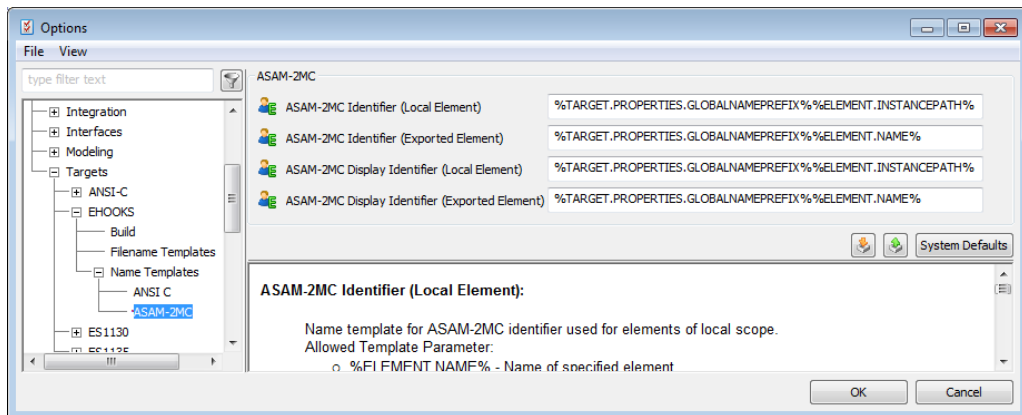

Figure 3.6: ASAM-2MC Name Options for the EHOOKS target

Table 3.1 on the next page lists the template parameters that can be used to configure ASAM-MCD-2MC names.

Template Parameter	Meaning	Remarks
`%ELEMENT.NAME%`	name of the specified element	
`%ELEMENT.INSTANCEPATH%`	instance path of element; starts at the element, up to the project (but excludes project) or an exported element	`%ELEMENT.INSTANCEPATH.FROM_ELEMENT%` identical to `%ELEMENT.INSTANCEPATH%`
`%ELEMENT.INSTANCEPATH.FROM_ELEMENT%`		

Template Parameter	Meaning	Remarks
`%ELEMENT.INSTANCEPATH.FROM_ELEMENT_TO_PROJECT%`	instance path of element; starts at the element, up to and including the project or an exported element	
`%ELEMENT.INSTANCEPATH.FROM_PROJECT_TO_ELEMENT%`	inverse instance path of element; starts at and includes the project or an exported element, down to the element	
`%ELEMENT.INSTANCEPATH.TO_ELEMENT%`	inverse instance path of element; starts at the project (but excludes project) or an exported element, down to the element	
`%COMPONENT.NAME%`	name of the associated ASCET component	
`%COMPONENT.IMPL.NAME%`	name of associated implementation set from the ASCET component	
`%TARGET.PROPERTIES.GLOBALNAMEPREFIX%`	name prefix specified in the ASCET options window, "Targets\ EHOOKS\Name Templates" node	see also section "Global Name Space Prefix" on page 22

Table 3.1: Template parameters for the configuration of ASAM-MCD-2MC names

Figure 3.7 on the next page shows the "Outline" tab for a project named `Proj_names`. The project uses an implementation named `Impl`.

The highlighted elements – the exported parameter `Ki` in the `Integrator` class included in the `IdleCon` module and the local variable `ndiff` in the `IdleCon` module – are used to determine the results of the ASAM-MCD-2MC name templates.
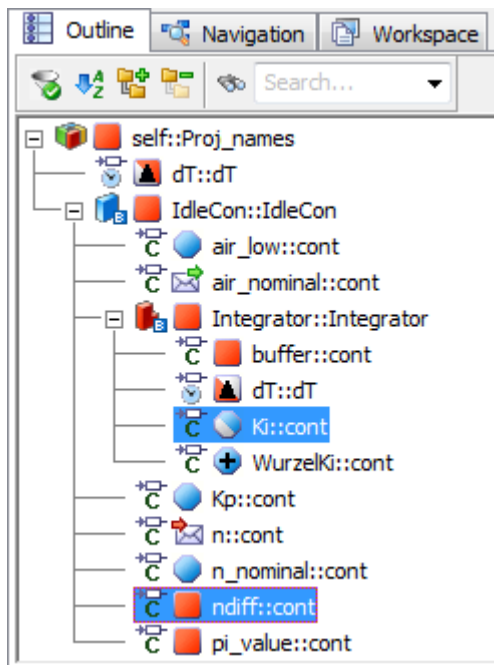
Figure 3.7: Example project for testing the ASAM-2MC name options

Template Parameter	Result for `ndiff`	Result for `Ki`
`%ELEMENT.NAME%`	`ndiff`	`Ki`
`%ELEMENT.INSTANCEPATH%` `%ELEMENT.INSTANCEPATH.` `FROM_ELEMENT%`	`ndiff.IdleCon`	`Ki`
`%ELEMENT.INSTANCEPATH.` `FROM_ELEMENT_TO_PROJECT%`	`ndiff.IdleCon.Proj_names`	`Ki`
`%ELEMENT.INSTANCEPATH.` `FROM_PROJECT_TO_ELEMENT%`	`Proj_names.IdleCon.ndiff`	`Ki`
`%ELEMENT.INSTANCEPATH.` `TO_ELEMENT%`	`IdleCon.ndiff`	`Ki`
`%COMPONENT.NAME%` `_%ELEMENT.NAME%`	**`IdleCon`**`_ndiff`	**`IdleCon`**`_Ki`
`%COMPONENT.IMPL.NAME%` `_%ELEMENT.NAME%`	**`Impl`**`_ndiff`	**`Impl`**`_Ki`

Table 3.2: Results of template parameters for the configuration of ASAM-MCD-2MC names

> **ℹ Note**
>
> Using `%COMPONENT.NAME%` or `%COMPONENT.IMPL.NAME%` without another template parameter that contains the element name leads to name clashes.

3.1.3.4 Cont Implementation Type

The code generator (`Object Based Controller *`; see
"To define an EHOOKS project" on page 18), in combination with the EHOOKS
target option "Cont Implementation Type" (see Figure 3.4 on page 21),
controls how ASCET generates bypass function code for continuous (real
number) elements in the model.

The following combinations are available:

Code Generator	Cont Implemen-tation Type	effect
`Object Based Controller Implementation`	`*`	Use the implementations specified in the model. When a variable is read from the ECU, the EHOOKS Target will automatically convert the value to the type defined in the model. When a variable is written to the ECU, the EHOOKS Target will automatically convert the value to the type used by the ECU.
`Object Based Controller Physical`	`Phys. Single Precision`	Generate all continuous elements as single-precision floating-point values. When a variable is read from the ECU the EHOOKS Target will automatically convert the value to single-precision floating-point. When a variable is written to the ECU the EHOOKS Target will automatically re-quantize the value to use the quantization defined by the ECU.
	`Phys. Double Precision`	Generate all continuous elements as double-precision floating-point values. When a variable is read from the ECU the EHOOKS Target will automatically convert the value to double-precision floating-point. When a variable is written to the ECU the EHOOKS Target will automatically re-quantize the value to use the quantization defined by the ECU.

Table 3.3: Effects of "Code Generator" and "Cont Implementation Type" combinations

3.1.4 Basic EHOOKS Configuration

If you decided to create a new EHOOKS `*.ehcfg` configuration file, then you need to start EHOOKS and configure the locations of the `*.hex` and `*.a2l` files.
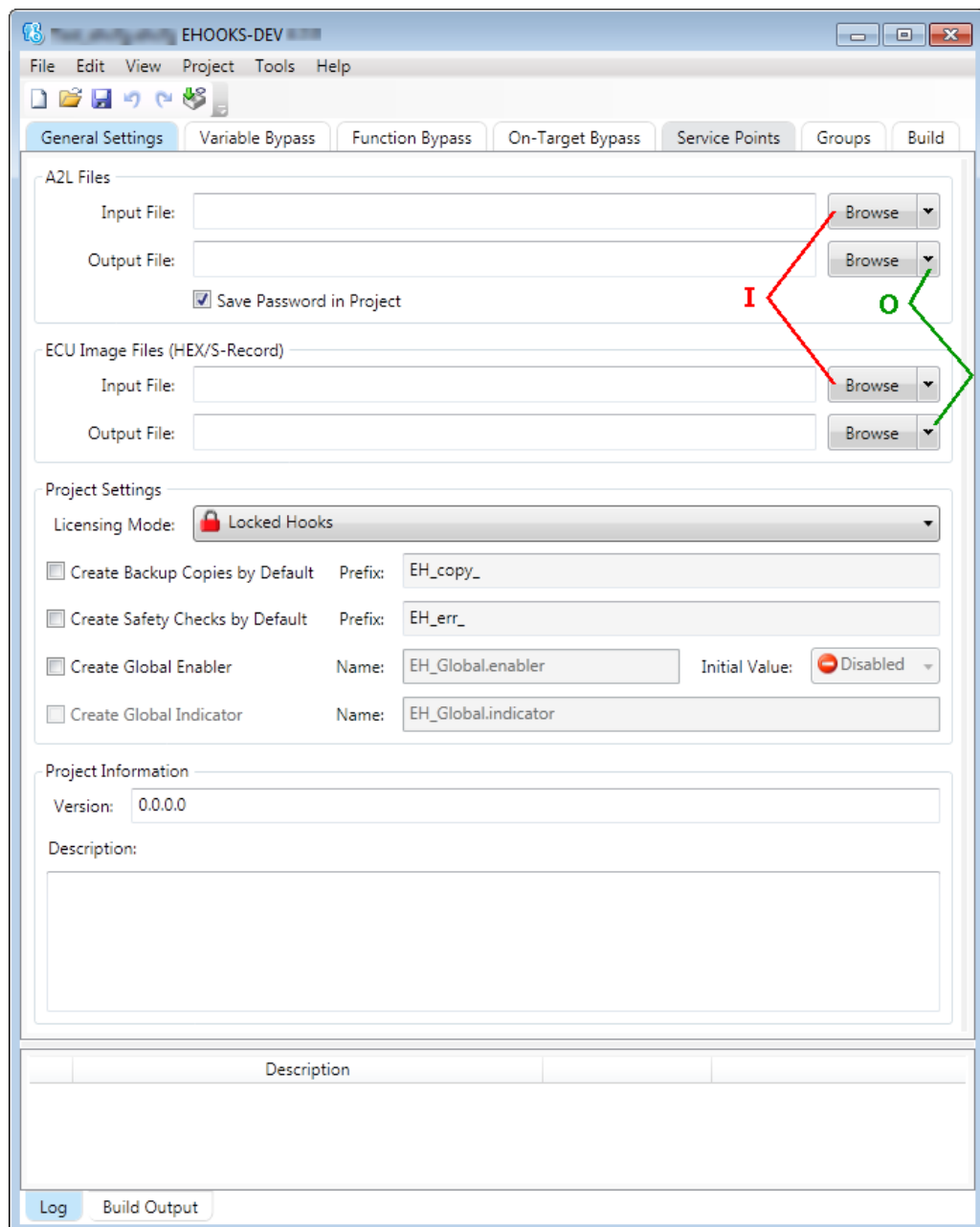

Figure 3.8: EHOOKS-DEV window: Choosing EHOOKS files (I: input files, O: output files)

To configure input and output files

1. In the "EHOOKS" tab, click on **Start EHOOKS** (see Figure 3.3 on page 20).

   If EHOOKS is not running, it is started now. The `*.ehcfg` file is opened in the EHOOKS-DEV window.

2. In the EHOOKS-DEV window, use the first and third **Browse** buttons (I in Figure 3.8 on the previous page) to select input `*.a2l` and `*.hex` files.

   If you access a password-protected `*.a2l` file for the first time, you are asked for a password.

3. Enter the password and click **OK**.

4. Activate the **Save Password in Project** option to store the password in the `*.ehcfg` file.

5. In the EHOOKS-DEV window, use the second and fourth **Browse** buttons (O in Figure 3.8 on the previous page) to enter output `*.a2l` and `*.hex` files.

6. In the EHOOKS-DEV window, select **File > Save** to save the `*.ehcfg` file.

ASCET will generate the warning shown in Figure 3.9 on this page if you do not specify any files in EHOOKS.
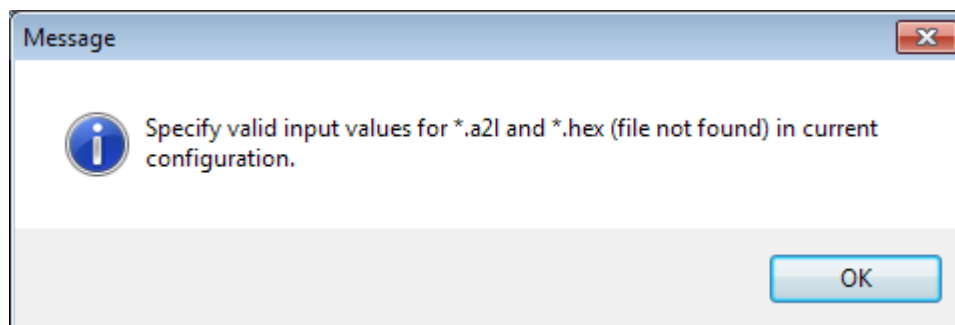

Figure 3.9: Warning if no EHOOKS files are selected

3.2   Integrating Bypass Functions

Bypass functions are created as normal ASCET models, and integrated in an ASCET project in the same way as any other ASCET model. Please consult the ASCET online help if you are unsure about how to create ASCET models.

The project can be an arbitrarily complex ASCET model.[1]

---

[1] Not all functionality is currently supported. See the ASCET-SE release notes for known limitations in this release.

3.2.1   Preparing the Project

When you integrate a normal ASCET project for series production, the code generator checks that:

- every sent message has a receiver
- every received message has a sender

ASCET will generate warnings if these checks fail.

When you build a bypass function, however, your model will typically have "unconnected" messages because they will be sent from or received by the ECU.

ASCET needs to know that these "loose ends" will be joined up. You can do this by selecting **Extras > Resolve Globals** in the project editor, as shown in Figure 3.10 on this page. In the EHOOKS Target this creates "virtual" messages that can then be hooked onto ECU variables.
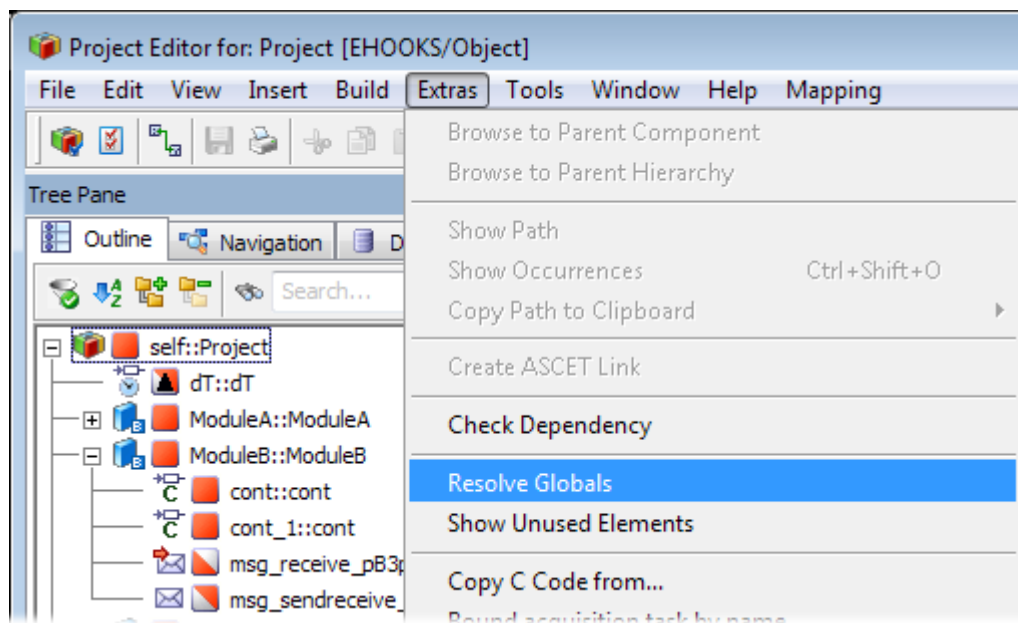
Figure 3.10: Resolving globals

3.2.2   Connecting Inputs and Outputs to ECU Variables

Messages that are sent or received by the project technically have no sender or receiver as the project context is passive. These messages represent the *unconnected* parts of the ASCET model. To connect them to ECU variables, you need to do the following things:

A. select ECU measurements and ECU write hooks

B. map messages that have no sender to ECU measurements (the message will be *read* from the ECU)

C. map messages that have no receiver to ECU write hooks (the messages will be *written* to the ECU)

The "Input" and "Output" sub-tabs of the "EHOOKS" tab in the ASCET project editor allow mapping ASCET messages to ECU variables.

> **ⓘ Note**
>
> To make sure that the view in these tabs is up to date, click the **Update** button to refresh the tab.

The "Input" and "Output" tabs are described in section "Input" Tab on the next page and section "Output" Tab on page 41. Section "Mapping Messages and ECU Variables" on page 44 contains detailed instructions for manual mapping, and section "Auto-Mapping" on page 53 describes automatic mapping.

3.2.2.1 "Input" Tab
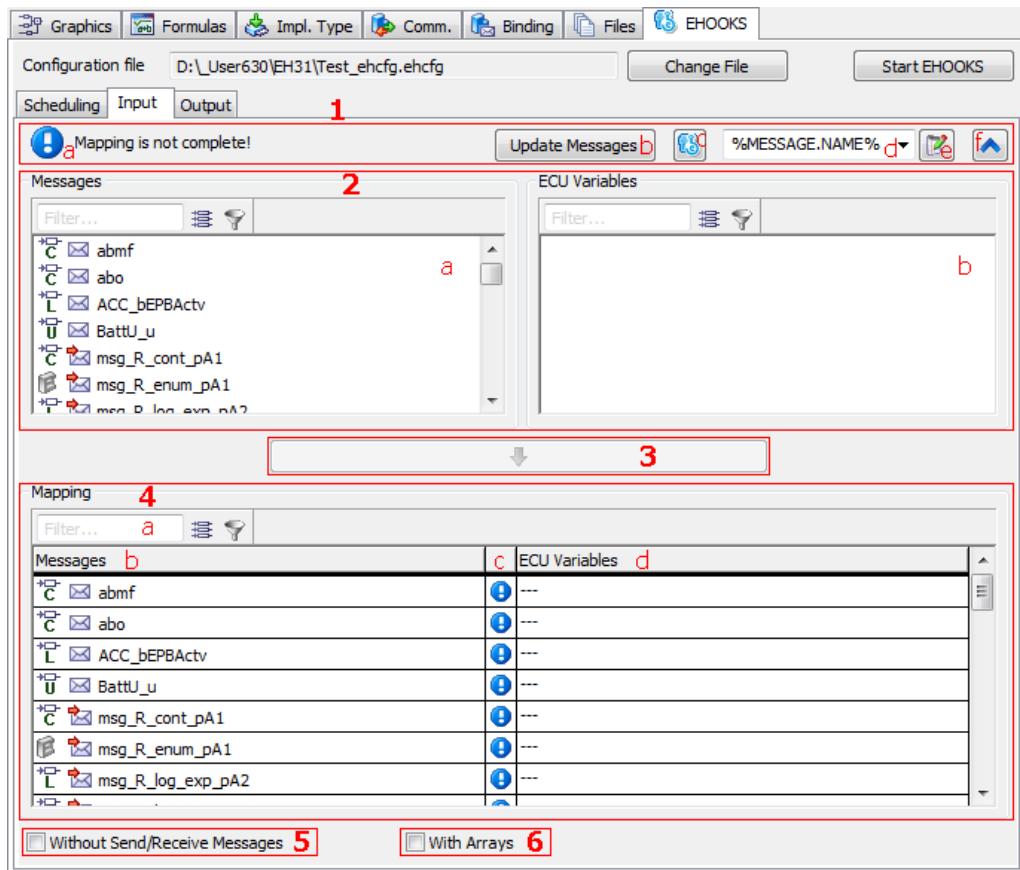
Figure 3.11: "Input" tab

The "Input" tab (shown in Figure 3.11 on this page) contains the following GUI elements:

A. top bar

    (a) information field

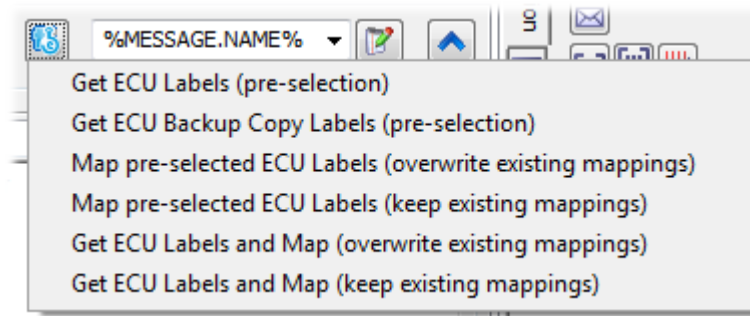   Shows whether message mapping is complete (✓), incomplete (ℹ), or contains invalid mappings (✗).

    (b) **Update Messages** button

        Updates the instances of the modules, i.e. imports changes in these components into the project.

    (c) **Open EHOOKS functions** button

        This button opens a dropdown menu with the following functions:

(d) combo box for auto-mapping pattern

In this combo box, you determine the pattern used for automatic mapping. See section 3.2.2.4 "Auto-Mapping" on page 53 for details.

(e) button

Opens the ASCET options window in the "Appearance\Editors\Project\Ehooks" node. In that node, you can adjust the patterns available for automatic mapping.

(f) / button

Shows () or hides () the upper table.

B. upper table (hidden by default)

(a) "Messages" column

This column lists all receive and send&receive messages from all modules directly or indirectly used in the project. The messages are displayed as follows:

scalar messages	exported/ imported	*<message>*
	local	`self.`*<module_inst>*[2] `(.`*<nested module_inst>*`...)`[3]`.`*<message>*
messages of array type	exported/ imported	*<message>*`[0..`*<n>*[4]`]`
	local	`self.`*<module_inst>* `(.`*<nested module_inst>*`...)` `.`*<message>*`[0..`*<n>*`]`
messages of record type	exported/ imported	*<record_inst>*[5]`(.`*<nested record_inst>*`...)` `.`*<record element>*
	local	`self.`*<module_inst>* `(.`*<nested module_inst>*`...)` `.`*<record_inst>*`(.`*<nested record_inst>*`...)` `.`*<record element>*

> ### ℹ️ **Note**
>
> Receive messages with an external Set method are **not** shown.
>
> Messages of array type are shown if **With Arrays** is activated.
>
> Messages of matrix type are **not** shown.
>
> Messages of record type are **not** shown directly. The record fields are shown instead.
>
> If one module contains a send message `<name>`, and another module contains a receive message with identical `<name>`, the receive message `<name>` is **not** shown.

∗   input field and 🔽 button above the column

You can enter a text string in the input field and then click on 🔽 to filter the list of available messages by name. The filter is case-insensitive; it finds all messages whose names contain the text string.

∗   ≣ properties filter above the column

Opens the "Filter Criteria" dialog window (see Figure 3.20 on page 52), which allows filtering the list by selected properties.

---

[2] `<module_inst>` is the module instance name
[3] optional for nested modules or records
[4] `<n>` is array_size - 1
[5] `<record_inst>` is the record instance name

An active type filter is indicated by a green overlay icon on both filter buttons:

An active filter is indicated by a green overlay icon: Click the button to remove the filter.

(b) "ECU Variables" column

This column lists all unmapped ECU measurement variables that are available for mapping. The elements are displayed as follows:

scalar ECU variables		*<ECU variable>*
ECU variables of array type	**With Arrays** activated	*<ECU array>*[0..*<n>* [6]]
	With Arrays deactivated	*<ECU array>*[*<i>*[7]]

<center>(A)   (B)</center>
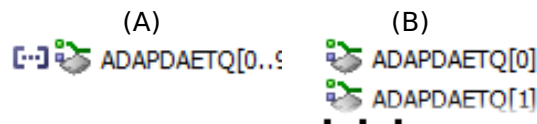
ECU variables of array type — (A:) **With Arrays** is activated; (B): **With Arrays** is deactivated

* input field, properties filter and button above the column

    The same as in the "Messages" column.

(c) context menu

* **Get ECU Labels (pre-selection)**

    Opens the EHOOKS variable selection dialog window (see Figure 3.16 on page 45). Variables you select there will be available in the "ECU Variables" columns when you close the window with **OK**.

    See also To select ECU variables on page 44.

* **Get ECU Backup Copy Labels (pre-selection)**

    Opens the EHOOKS backup copy selection dialog window (see Figure 3.18 on page 47). This allows to map a message to a backup copy of an ECU write hook (i.e. the value calculated by the original ECU before the ECU variable was hooked and bypassed by EHOOKS).

    See also To use the **Get ECU Labels and Map (*)** commands on page 59.

---

[6] `<n>` is array_size - 1

[7] `<i>` is the number of the array element (0 .. array_size - 1)

   ∗ **Map pre-selected ECU Labels (overwrite existing map-pings)** and **Map pre-selected ECU Labels (keep existing mappings)**

Variables in the "ECU Variables" field are mapped automatically to messages with matching names. Existing mappings are treated according to the function you selected.

See also To use the **Map pre-selected ECU Labels (*) commands** on page 56.

   ∗ **Get ECU Labels and Map (overwrite existing mappings)** and **Get ECU Labels and Map (keep existing mappings)**

The ECU variables are searched for variables that match ASCET message names. Detected matches are mapped automatically. Existing mappings are treated according to the function you selected.

See also To use the **Get ECU Labels and Map (*) commands** on page 59.

C. [⬇] button

Maps a message selected in the "Message" column to an ECU variable selected in the "ECU Variables" column.

> **ℹ Note**
>
> In the "Input" tab, one message can be mapped to one ECU variable. However, you can map several messages to the same ECU variable.

D. "Mapping" field - lower table

(a) input field, ☰ properties filter and ▽ button

The same as in the "Messages" column of the upper table; see page 34.

(b) "Messages" column

This column lists the same messages as the "Messages" column in the upper table; see the description on page 33.

(c) icon column

This column contains icons that represent the mapping status.

ℹ	The message is unmapped.
✅	Mapping is valid: the message is mapped to a suitable ECU variable
❌	Mapping is invalid

(d) "ECU Variables" column

This column shows the ECU variables mapped to the messages in the "Messages" column of the "Mapping" field. The elements are displayed as in the upper "ECU Variables" column.

If no mapping exists (---; see Figure 3.12 on the next page, 3rd row), the "ECU Variables" column can be used to perform

mapping. A double-click in a table cell opens a list of all suitable ECU variables (see Figure 3.12, 4th row).

Unsaved changed mappings appear in blue font (see Figure 3.12, 2nd row).

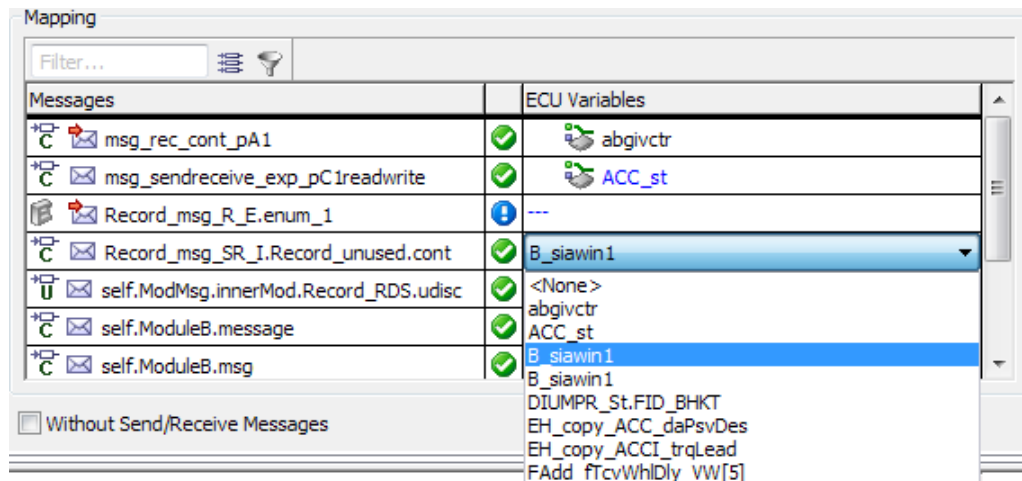

Figure 3.12: "Mapping" field in the "Input" tab

    (e) context menu

         ∗ **Edit**

            Opens the list of available ECU variables for selection.

         ∗ **Remove**

            Removes the selected mapping.

         ∗ **Remove All**

            Removes all existing mappings.

         ∗ **Revert Changes**

            Reverts unsaved mapping changes.

> **ℹ Note**
>
> **Edit**, **Remove**, **Remove All**, and **Revert Changes** work the same way as the respective commands described in the ASCET online help for message and parameter mapping in AUTOSAR software components.

         ∗ **Get ECU Labels (pre-selection)**

            The same as in the context menu of the upper table; see page 35.

            See also To select ECU variables on page 44.

         ∗ **Get ECU Backup Copy Labels (pre-selection)**

            The same as in the context menu of the upper table; see page 35.

See also
[To connect a message to a backup copy of an ECU variable](#) on page 45.

∗ **Map pre-selected ECU Labels (overwrite existing mappings)** and **Map pre-selected ECU Labels (keep existing mappings)**

The same as in the context menu of the upper table; see page 36.

See also

[To use the **Map pre-selected ECU Labels (*)** commands](#) on page 56.

∗ **Get ECU Labels and Map (overwrite existing mappings)** and **Get ECU Labels and Map (keep existing mappings)**

The same as in the context menu of the upper table; see page 36.

See also [To use the **Get ECU Labels and Map (*)** commands](#) on page 59.

∗ **Export**

Opens the "Export Settings" dialog window where you can export mappings to an `*.xml` or `*.csv` file.

∗ **Import**

Imports mappings from an `*.xml` or `*.csv` file.

> ### ℹ️ **Note**
>
> Instructions for **Export** and **Import** are given in section 3.2.4 "[Exporting and Importing Mappings](#)" on page 65.

∗ **Create ASCET Link**

Creates an ASCET link for the selected source or target of a mapping, or the selected entire mapping, in the "Mapping" field. The link opens the project and highlights the selected mapping part or mapping in the appropriate sub-tab of the "EHOOKS" tab.

See the ASCET online help for further information on ASCET links.

The **Mapping** menu contains the same options as the context menu in the lower table.

E. **Without Send/Receive Messages** option

If activated, no send&receive messages appear in the upper and lower "Messages" columns of the "Inputs" tab.

> **ℹ Note**
>
> Do not deactivate this option after you have mapped send&receive messages. If you do, the existing mappings become invalid.

The state of this option is stored when the project editor is closed.

F. **With Arrays** option

If the option is activated (A in Figure 3.13 on the next page), messages of array type appear in the upper and lower "Messages" columns of both "Input" and "Output" tab. ECU arrays are listed as arrays in the "ECU Variables" columns.

If the option is deactivated (B in Figure 3.13 on the next page), array messages do not appear in the "Messages"column; they cannot be mapped. Each element of an ECU array is listed separately in the "ECU Variables" columns.

> **ℹ Note**
>
> An ECU variable appearing as an array element may either actually be an array element, or a regular element with an index as part of the label name. In the latter case, the ECU variable does respond to changes of the option.

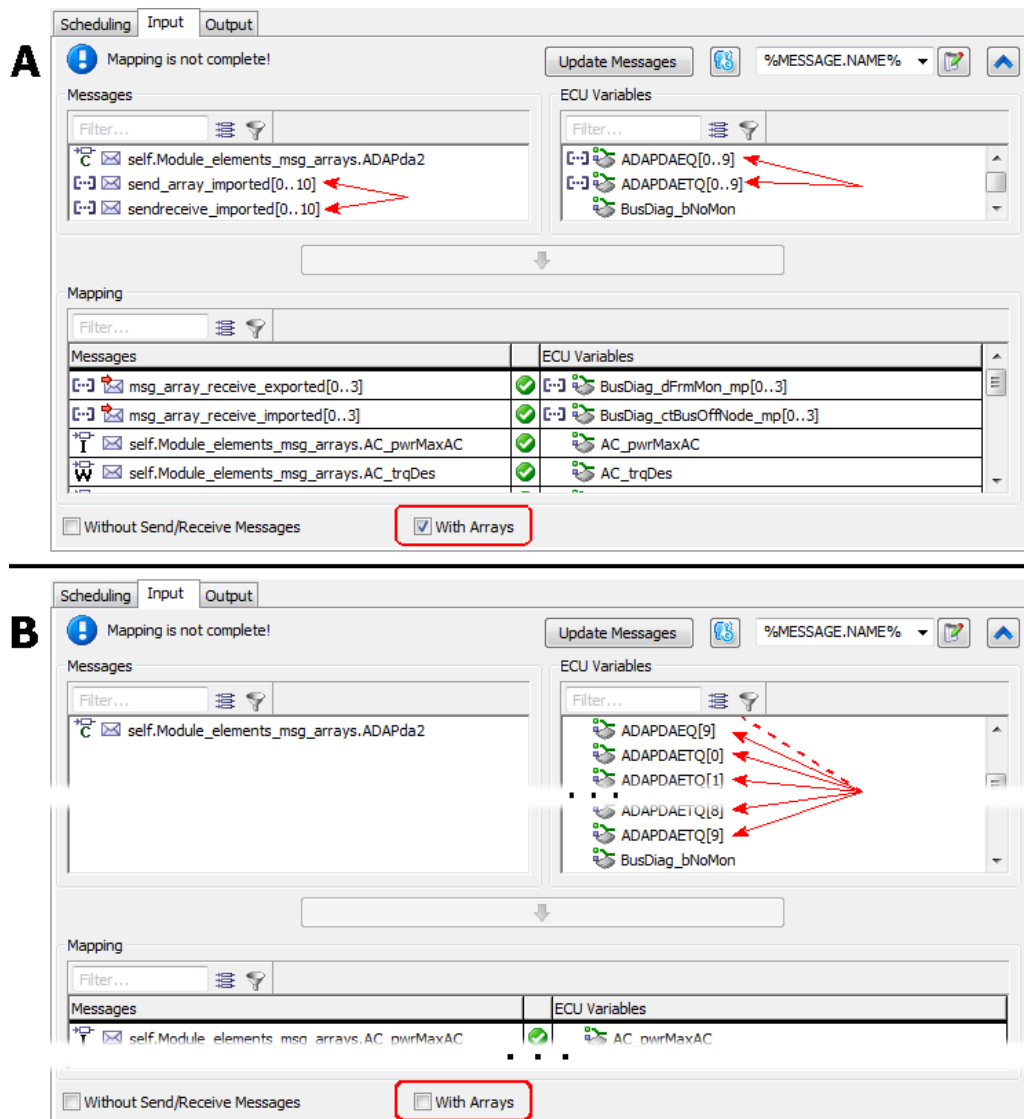The state of this option is stored when the project editor is closed.

Figure 3.13: "Input" tab with (A) and without (B) activated **With Arrays** option
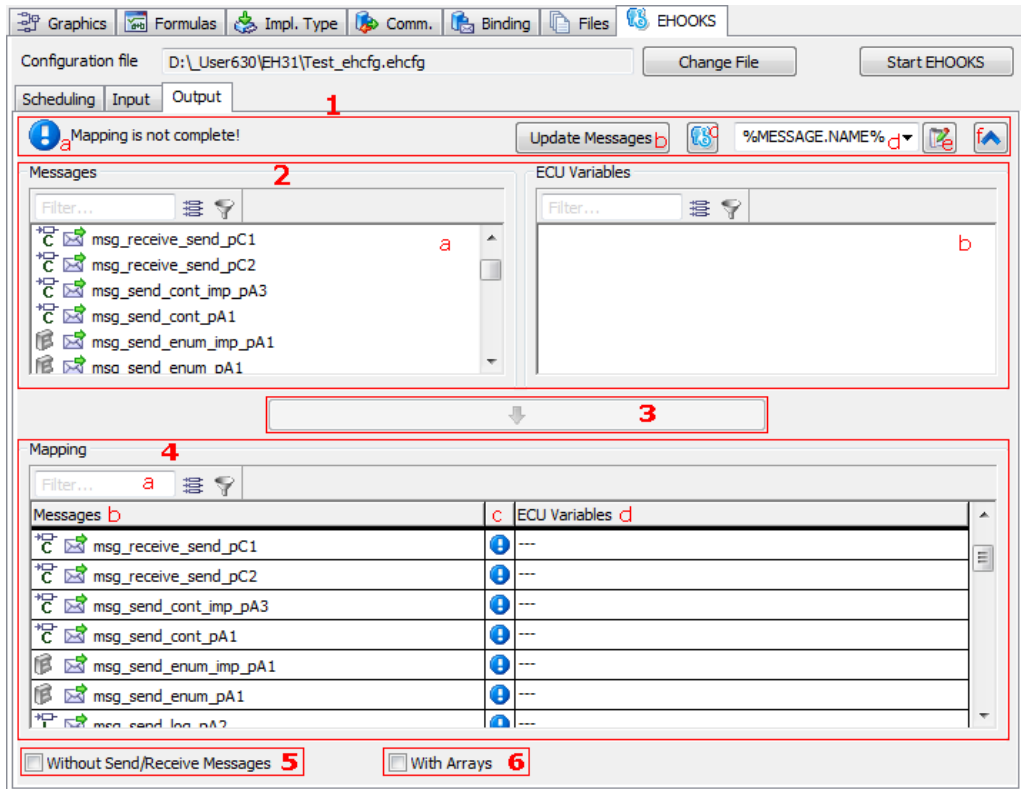
3.2.2.2 "Output" Tab

Figure 3.14: "Output" tab

The "Output" tab contains the following GUI elements:

A. top bar

Contains the same elements as the top bar in the "Input" tab; see page 32.

B. upper table (hidden by default)

(a) "Messages" column
This column lists all send and send&receive messages from all modules directly or indirectly used in the project. The messages are displayed as follows:

scalar messages	exported/ imported	*<message>*
	local	`self.`*<module_inst>*[8] `(.`*<nested module_inst>*`...)* [9] `.`*<message>*
messages of array type	exported/ imported	*<message>*`[0..`*<n>* [10]`]`
	local	`self.`*<module_inst>* `(.`*<nested module_inst>*`...)* `.`*<message>* `[0..`*<n>*`]`
messages of record type	exported/ imported	*<record_inst>*[11]`(.`*<nested record_inst>*`...)* `.`*<record element>*
	local	`self.`*<module_inst>* `(.`*<nested module_inst>*`...)* `.`*<record_inst>* `(.`*<nested record_inst>*`...)* `.`*<record element>*

> ### ℹ Note
>
> Send messages with an external Get method are shown.
>
> Messages of array type are shown.
>
> Messages of matrix type are **not** shown.
>
> Messages of record type are **not** shown directly. The record fields are shown instead.
>
> If one module contains a send message `<name>`, and another module contains a receive message with identical `<name>`, the send message `<name>` is shown.

* input field, 葺 properties filter and 🢒 button above the column
  The same as in the "Input" tab, "Messages" column, of the upper table; see the description on page 34.

(b) "ECU Variables" column
This column lists all unmapped ECU Write Hook variables that are available for mapping. The elements are displayed the same way as in the "Input" tab; see the description on page 35.

---

[8] `<module_inst>` is the module instance name
[9] optional for nested modules or records
[10] `<n>` is array_size - 1
[11] `<record_inst>` is the record instance name

* input field, ☰ properties filter and ▽ button above the column

   The same as in the "Input" tab, "Messages" column, of the upper table; see the description on page 34.

(c) context menu

   The same as the context menu in the upper table of the "Input" tab (see page 35), except that **Get ECU Backup Copy Labels (pre-selection)** is deactivated.

C. ⬇ button

   Maps a message selected in the "Message" column to an ECU variable selected in the "Variables" column.

   > **i Note**
   >
   > In the " Output" tab, one message can be mapped to several ECU variables.

D. "Mapping" field - lower table

(a) input field, ☰ properties filter and ▽ button

   The same as in the "Input" tab, "Messages" column, of the upper table; see the description on page 34.

(b) "Messages" column

   This column lists the same messages as the "Messages" column in the upper table; see page 42. If a message is mapped more than once, each mapping is shown in a separate row.

(c) icon column

   The same as in the "Input" tab; see the description on page 36.

(d) "ECU Variables" column

   This column shows the ECU variables mapped to the messages in the "Messages" column of the "Mapping" field. The elements are displayed as in the upper "ECU Variables" column.

   If no mapping exists (---; see Figure 3.15 on the next page, 3rd row), the "ECU Variables" column can be used to perform mapping. A double-click in a table cell opens a list of all suitable ECU variables (see Figure 3.15, 4th row).

   Unsaved changed mappings appear in blue font (see Figure 3.15, 2nd row).

(e) context menu

   The same as the context menu in the lower table of the "Input" tab (see page 37), except that **Get ECU Backup Copy Labels** is deactivated.

Figure 3.15: "Mapping" table in the "Output" tab

E. **Without Send/Receive Messages** option

If activated, no send&receive messages appear in the upper and lower "Messages" columns of the "Outputs" tab.

> **ℹ Note**
>
> Do not deactivate this option after you have mapped send&receive messages. If you do, the existing mappings become invalid.

The state of this option is stored when the project editor is closed.

F. **With Arrays** option

Works the same way as in the "Input" tab; see the description on page 38.

3.2.2.3 Mapping Messages and ECU Variables

This section contains step-by-step instructions for selecting ECU variables and mapping them to messages.

To select ECU variables

1. In the project editor, go to the "EHOOKS" tab.

2. Do one of the following:

   – To select ECU Measurement variables, go to the "Input" tab (Figure 3.11 on page 32).

   – To select ECU Write Hook variables, go to the "Output" tab (Figure 3.14 on page 41).

3. To open the EHOOKS variable selection dialog window (see Figure 3.16 on the next page), do one of the following:

   – Right-click in the tab and select **Get ECU Labels (pre-selection)** from the context menu.

   – Select **Mapping > Get ECU Labels (pre-selection)**.

– Click on the ![icon] **Open EHOOKS functions** button and
select **Get ECU Labels (pre-selection)**.

4. In the EHOOKS variable selection dialog window, select the
required EHOOKS variables, then click **OK**.

The selected ECU variables appear in the "ECU Variables"
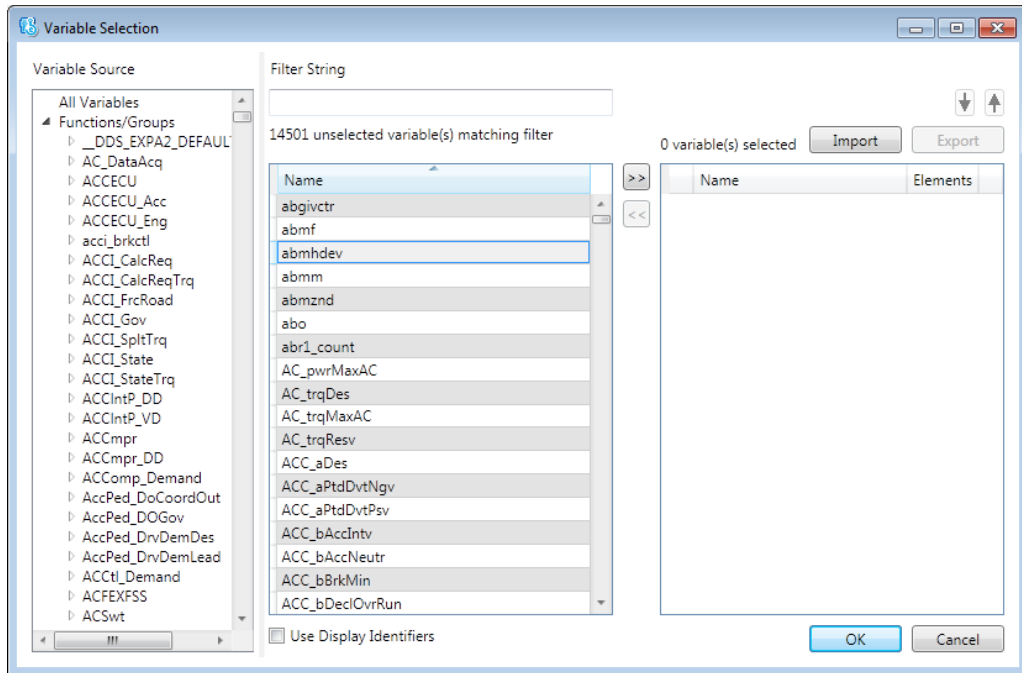column in the upper table of the "Input" or "Output" tab.

Figure 3.16: EHOOKS variable selection dialog window
        (see the EHOOKS-DEV user guide for further information)

ℹ **Note**

It is recommended that you leave the EHOOKS option **Convert All** acti-
vated. This will cause EHOOKS to generate the conversion functions from
ECU types to floating-point types. ASCET uses these functions when gener-
ating code for the `Object Based Controller Physical` code genera-
tor (see section 3.1.3.4).

To connect a message to a backup copy of an ECU variable

ℹ **Note**

Backup copies are only available for ECU measurement variables whose
"Create Backup Copy" property is set to `Yes`; see Figure 3.17 on the next
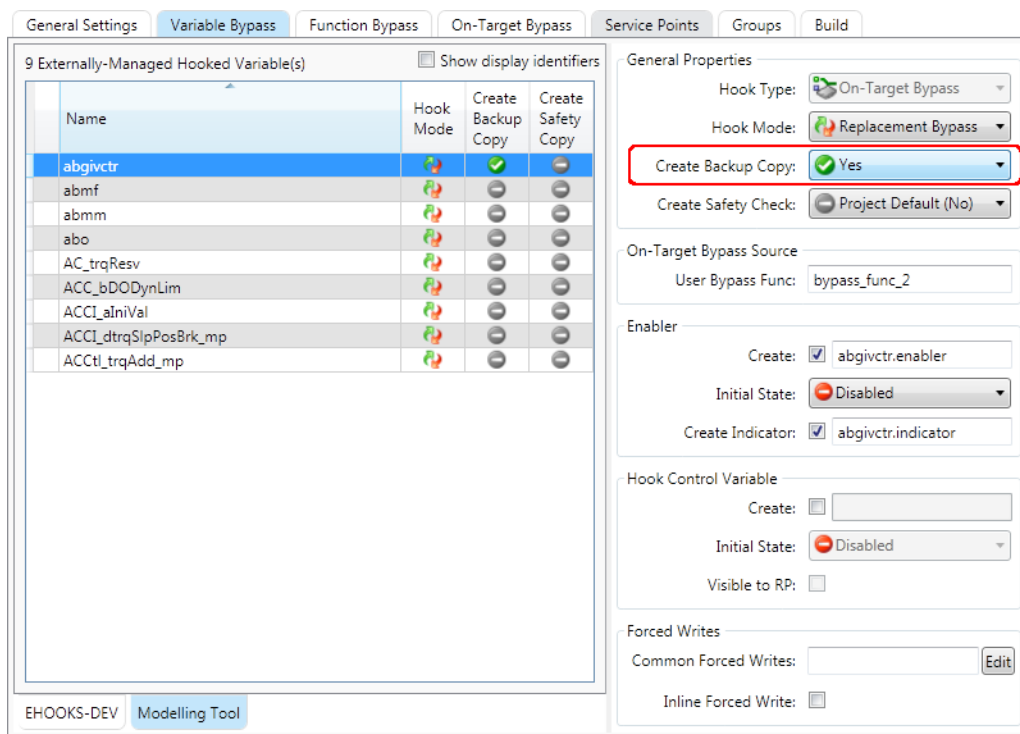page.

Figure 3.17: Activating backup copies in the "Variable Bypass" tab of the EHOOKS window (see the EHOOKS-DEV user guide for further information)

1. In the project editor, go to the "Input" sub-tab of the "EHOOKS" tab.

2. Do one of the following:

   – Right-click in the tab and select **Get ECU Backup Copy Labels (pre-selection)** from the context menu.

   – Select **Mapping > Get ECU Backup Copy Labels (pre-selection)**.

   – Click on the **Open EHOOHS functions** button and select **Get ECU Backup Copy Labels (pre-selection)**.

   The "Hook Selection" window (see Figure 3.18 on the next page) opens. The left table lists all ECU variables with backup copy enabled.

3. In the left column of the "Hook Selection" window, select the EHOOKS variables whose backup copies you want to connect to ASCET messages.

4. Click the button to shift the selected ECU variables to the right column.

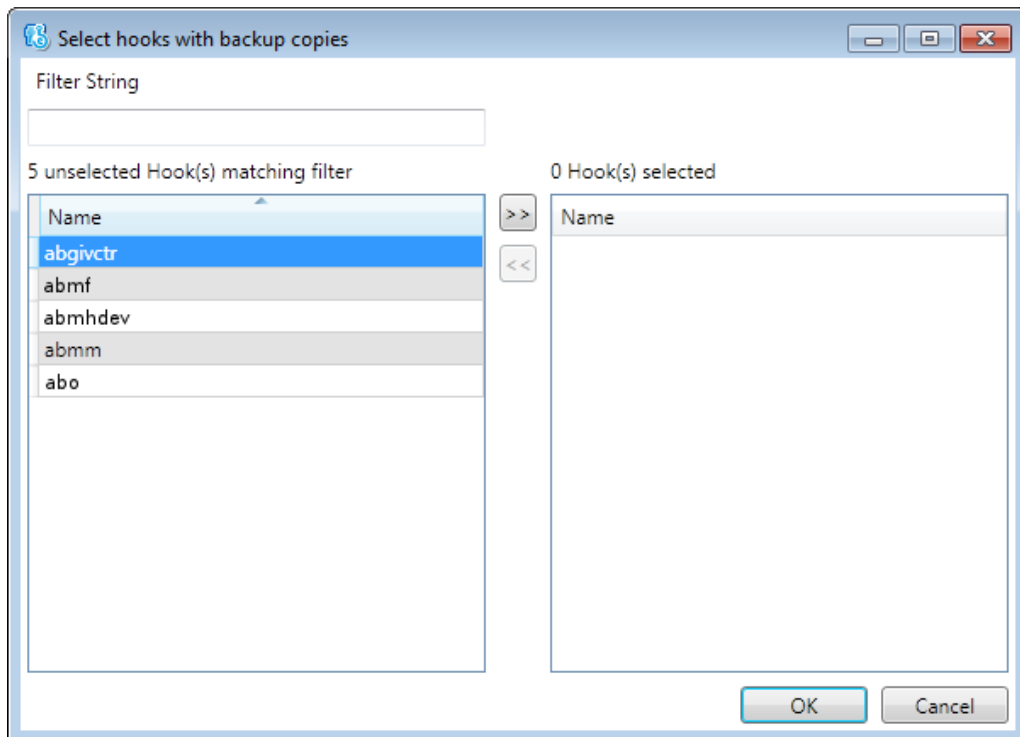5. Click **OK** to close the "Hook Selection" window.

Figure 3.18: EHOOKS "Hook Selection" window

Backup copies (named `EH_copy_<ecu variable>`) of the selected ECU variables are now available for mapping (see Figure 3.19 on page 48).

6. Map the backup copies to ASCET messages as described in "To map messages and ECU variables in the "Mapping" field on page 49" or "To map messages and ECU variables in the upper table" on page 50.

ℹ️ Note

Note that when selecting a backup copy, the GUI presented by EHOOKS supports multiple selection. ASCET can only use a *single* selection. If you select more than one backup variable per message using the dialog, ASCET will only use the first item you select.
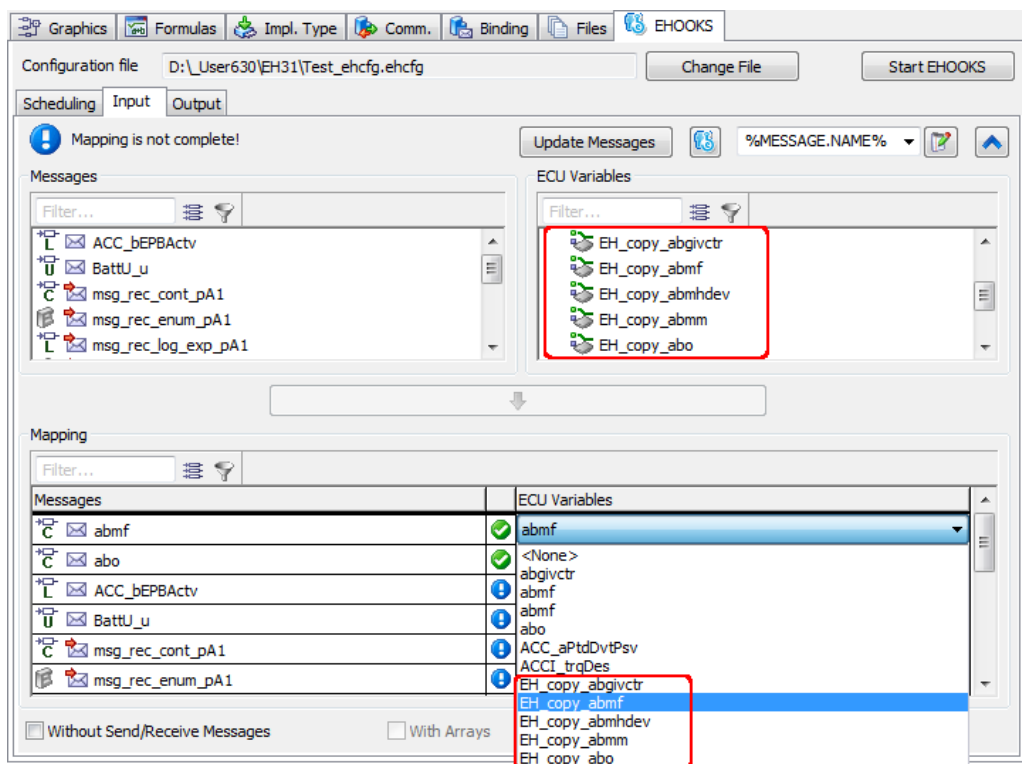

Figure 3.19: Backup copies of ECU measurement variables available for
mapping

To map messages and ECU variables in the "Mapping" field

> ℹ️ **Note**
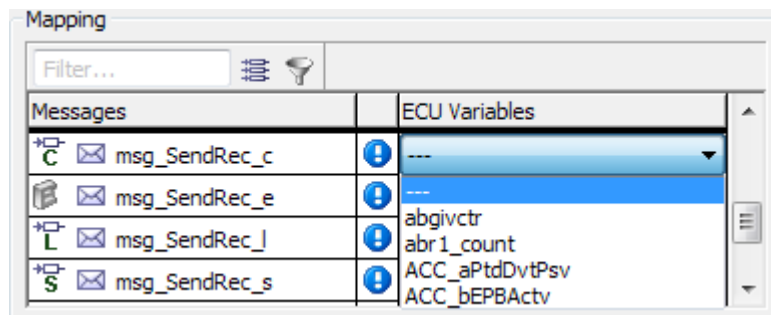>
> You cannot use the "Mapping" field in the "Output" tab for multiple mappings of the same message.

This instruction applies to scalar messages and scalar elements of record messages.

1. In the project editor, go to the "EHOOKS" tab.

2. Do one of the following:

   – To map ECU Measurement variables, go to the "Input" tab (Figure 3.11 on page 32).

   – To map ECU Write Hook variables, go to the "Output" tab (Figure 3.14 on page 41).

3. If desired, filter the columns (see also "To filter the columns" on page 51).

4. In the "Mapping" field, double-click in a cell in the "ECU Variables" column.

   A list with all ECU variables available for mapping opens.

5. Select an ECU variable.

   The mapping is performed. The results are shown in the "Mapping" field.

   In the "Output" tab, the mapped ECU variable is removed from the "ECU Variables" column of the upper table.
   In the "Input" tab, the mapped message is removed from the "Messages" column of the upper table.

   Changed mappings are indicated by blue font in the "ECU Variables" column of the "Mapping" field.

   The icon column in the "Mapping" field shows the mapping status; see the description on page 36.

To map messages and ECU variables in the upper table

This instruction applies to scalar messages and scalar elements of record messages.

1. In the project editor, go to the "EHOOKS" tab.

2. Do one of the following:
   - To map ECU Measurement variables, go to the "Input" tab (Figure 3.11 on page 32).
   - To map ECU Write Hook variables, go to the "Output" tab (Figure 3.14 on page 41).

3. If necessary, click on to show the upper table.

4. If desired, filter the columns (see also "To filter the columns" on the next page).

5. In the "Messages" column of the upper table, select a message.

6. In the "ECU Variables" column of the upper table, select an ECU variable.

   The button becomes available if the selected elements can be mapped.

7. Click on .

> ℹ️ **Note**
>
> As an alternative to these steps, you can drag a message from the "Messages" column and drop it onto a suitable element in the "ECU Variables" column.

The mapping is performed. The results are shown in the "Mapping" field.

In the "Output" tab, the mapped ECU variable is removed from the "ECU Variables" column of the upper table.
In the "Input" tab, the mapped message is removed from the "Messages" column of the upper table.

Changed mappings are indicated by blue font in the "ECU Variables" column of the "Mapping" field.

The icon column in the "Mapping" field shows the mapping status; see page 36.

To remove a selected message/ECU variable mapping

This instruction applies to scalar messages and scalar elements of record messages.

1. In the project editor, go to the "EHOOKS" tab.

2. Go to the "Input" (Figure 3.11 on page 32) or "Output"
   (Figure 3.14 on page 41) tab.

3. In the "Mapping" field, "Messages" or "ECU Variables" column,
   select a mapped element.

4. Do one of the following:

   – Open the context menu or the **Mapping** menu and select
     Remove.

   – Press <Delete>.

   – In the "Mapping" field, double-click a cell in the "ECU
     Variables" column and select <None>.

   The mapping is removed. If it was the $1+n^{th}$ mapping of a Send
   or SendReceive message, the entire line is removed from the
   "Mapping" field.

   The ECU variable reappears in the upper table.

   If the message is a Receive message, it reappears in the upper
   table, too.

   Changed mappings are indicated by blue font in the "ECU
   Variables" column of the "Mapping" field.

   The icon in the icon column is reset to ⬤.

To remove all message/ECU variable mappings in a tab

1. In the project editor, go to the "EHOOKS" tab.

2. Go to the "Input" (Figure 3.11 on page 32) or "Output"
   (Figure 3.14 on page 41) tab.

3. Open the context menu or the **Mapping** menu and select
   Remove All.

   All mappings in the tab are removed. Messages and ECU
   variables reappear in the upper table.

   The icons in the icon column are reset to ⬤.

To filter the columns

You can filter the columns in the "Input" or "Output" tabs for more clearness.
You can filter for element names or for element properties.

1. To filter for element properties, do the following:

   i. In the column you want to filter, click on the ▤ button.
      The respective "Filter Criteria" dialog window (Figure 3.20
      on the next page) opens.

   ii. In the combo boxes of the "Filter Criteria" dialog window,
       select the properties you want to show in the column.

iii. Click **OK** to apply the filter.

Only elements with all of the selected properties are shown in the list. The active type filter is indicated by a green overlay icon on both filter buttons:
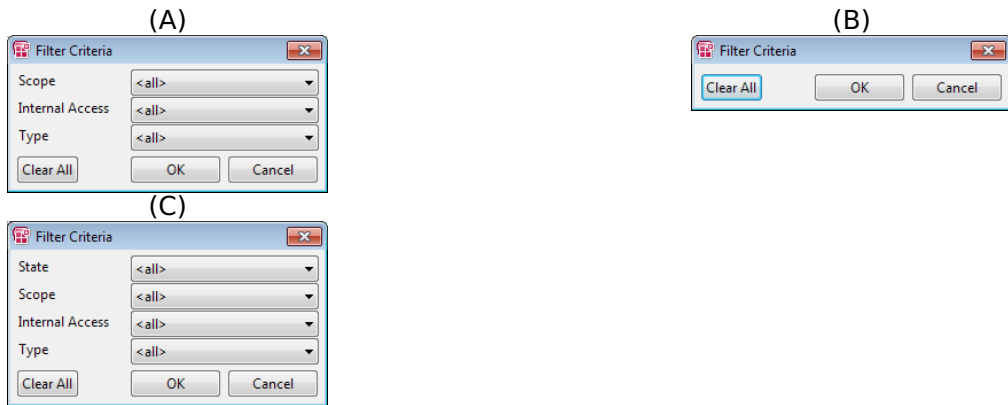
(A)

(B)

(C)

Figure 3.20: "Filter Criteria" windows

(A): upper table, "Messages" column; (B): upper table, "ECU Variables" column; (C): lower table

2. To filter for element name, do the following:

i. In the column you want to filter, enter a text string in the input field.

ii. Click on or press <Enter> to apply the filter.

Only elements whose names contain the text string are shown in the list. The filter is case-insensitive, i.e. a search term `Msg` will also find `msg`, `MSG`, etc. The active name filter is indicated by a green overlay icon on the second filter button:

In the "Mapping" field (lower table), the filter is applied to both columns (see also Figure 3.21 on this page). An entry is displayed if at least one name contains the text string.
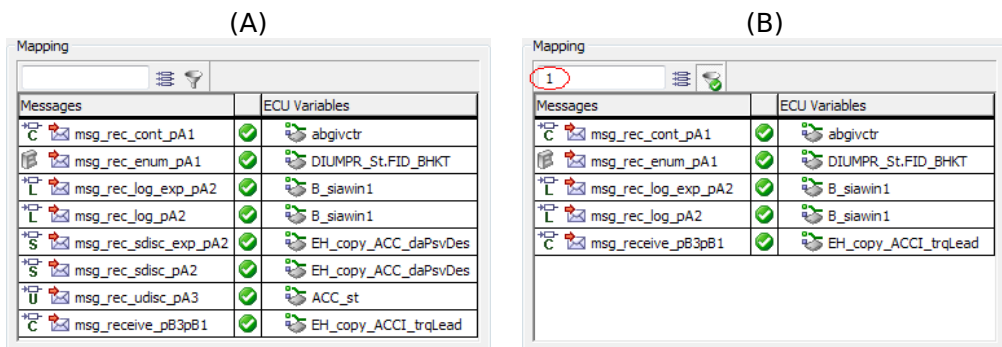
(A)

(B)

Figure 3.21: Example: Name filter in the "Mapping" field

(A): no name filter; (B): active name filter

3. If desired, combine both filters.

4. To deactivate all filters in a list, click on the 🔄 button of the respective list.

   The filter is deactivated, all entries of the respective list are shown. The filter settings, i.e. the text string in the input field and the settings in the "Filter Criteria" dialog window, are kept until you delete or overwrite them.

3.2.2.4 Auto-Mapping

Mapping each individual message can be time-consuming if you have a lot of variables. To simplify the task, the EHOOKS Target provides an *auto-mapping* function.

By default, auto-mapping maps messages and ECU measurements / write hooks with identical names. To make auto-mapping more effective, you can define patterns that are used to map messages and ECU measurements / write hooks with different names. With these patterns, you can, for example, take into account prefixes and/or postfixes.

Default patterns are defined in the ASCET options, "Appearance\Editors\Project\Ehooks" node (see Figure 3.22 on the next page). These patterns are available in the combo boxes of the "Input" and "Output" tabs; see (1d) in Figure 3.11 on page 32 and Figure 3.14 on page 41. You can add your own patterns, either via the ASCET options window or via the combo box.

Auto-mapping maps unconnected ASCET messages in the project to ECU measurements or write hooks with matching names according to the following heuristic:

- If a message has no sender (or is sent only by the project itself) and is received by one or more modules, then it will be automatically mapped to an ECU Measurement with a matching name.

- If a message has no receiver (or is received only by the project itself) and is sent by one module, then it will be automatically mapped to an ECU Write Hook with a matching name.

ECU variables are searched for matches in the following order:

A. ***identifier of the ECU variable***

   If an ECU variable with matching identifier is found, it is mapped to the ASCET message.

B. ***display identifier of the ECU variable***

   If no matching identifier is found, the display identifiers of the ECU variables are searched. If a matching display identifier is found, the respective ECU variable is mapped to the ASCET message.
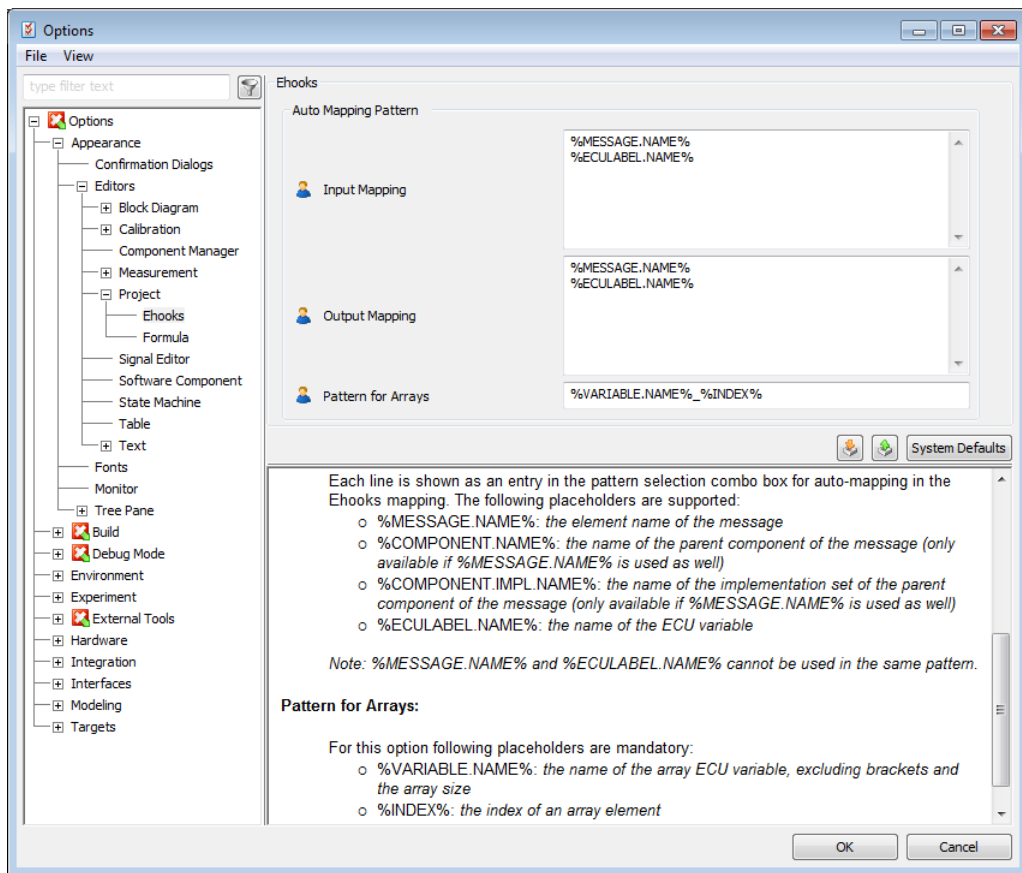
Figure 3.22: EHOOKS options: patterns for auto-mapping in the "Input" and
"Output" tabs

C. **an array element of an ECU array variable**

If no matching identifier and no matching display identifier are found, the names of ECU array variables are searched. If a matching array variable is found, the ASCET message is mapped to an element of this array.

The pattern for searching arrays is determined in the "Pattern for Arrays" field in the "Appearance\Editors\Project\Ehooks" node of the ASCET options window (see Figure 3.22 on the previous page.).

This pattern is used to find messages that can be mapped to array elements. If an ECU array can be found by applying the inferred label name to the mapping pattern (both considering the A2L identifier and display identifier), and the inferred index is within the range of the ECU array, then the message is mapped to the array element with that index.

If a mapping is done from ECU variables to messages (i.e. using patterns based on `%ECULABEL.NAME%`), he name of the ECU variable is first passed to the automapping pattern and then to the array pattern.

If the array pattern does not contain both the `%VARIABLE.NAME%` and `%INDEX%)` parameters it is considered invalid and no additional array element mapping will be done. This allows for disabling this auto-mapping functionality.

This procedure works for all auto-mapping commands (see Figure 3.23 on page 57).

> ### ℹ️ Note
>
> There is no guarantee that a message and an ECU variable with matching names represent the same concept.
>
> For example, a message named `Speed` in the model representing speed in km/h is not the same as a message named `Speed` on the ECU that represents speed in miles/h.
>
> You must therefore verify that any auto-mappings represent valid connections by using the ECU information provided by your ECU supplier.

Matching names are derived from the currently selected pattern. `%MESSAGE.NAME%` is the template for names of elements in the "Messages" column, `%ECULABEL.NAME%` is the template for names of elements in the "ECU Variables" column. If you add a prefix or postfix to a template, auto-mapping works as shown in Table 3.4 on the next page.

pattern	auto-mapping result
`prefix_%MESSAGE.NAME%`	A message `name` and an ECU variable `prefix_name` are mapped.

`%MESSAGE.NAME%_postfix`	A message `name` and an ECU variable `name_postfix` are mapped.
`prefix_%ECULABEL.NAME%`	A message `prefix_name` and an ECU variable `name` are mapped.
`%ECULABEL.NAME%_postfix`	A message `name_postfix` and an ECU variable `name` are mapped.

Table 3.4: Auto-mapping patterns and auto-mapping results

---

i **Note**

Keep in mind the following rules when you define a template:

- You can add a prefix, or a postfix, or both.

- Prefixes and postfixes are case-sensitive; a pattern `PREFIX_<name>` will not match a message or ECU variable `prefix_<name>`.

- You cannot use `%MESSAGE.NAME%` and `%ECULABEL.NAME%` in the same pattern.

Auto-mapping is accessed via the **Mapping** menu, the **Open EHOOKS functions** button or the context menu in the "Input" and "Input" tabs ((A) – (C) in Figure 3.23 on the next page).

Auto-mapping has the following modes:

Overwrite existing mappings replaces any mappings you have done with the mappings that are automatically detected.

Keep existing mappings adds automatically detected mappings only if a mapping is not already defined.

ASCET will show the changes that auto-mapping has made by highlighting the mappings in blue text. The highlighting is removed when you save the project.

To use the Map pre-selected ECU Labels (*) commands

1. Go to the "Input" or "Output" tab.

2. In the combo box[12], define or select a pattern for auto-mapping.

3. Do one of the following:

    – Select **Mapping > Map pre-selected ECU Labels (*** [13]**)**.

    – Click on the 🔧 **Open EHOOKS functions** button and select **Map pre-selected ECU Labels (*** [13]**)**.

---

[12] see (1d) in Figure 3.11 on page 32 and Figure 3.14 on page 41
[13] * is either **overwrite existing mappings** or **keep existing mappings**
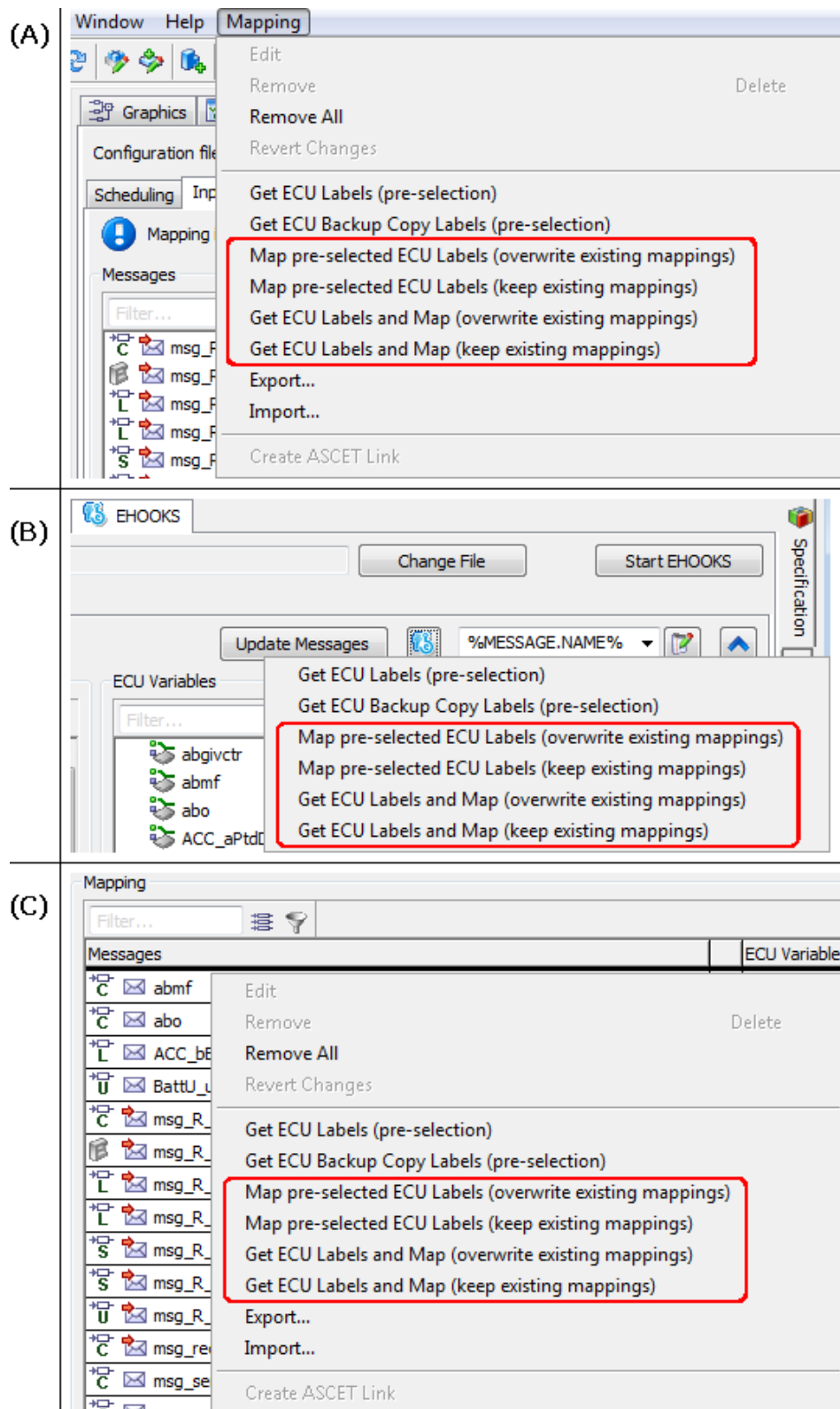
Figure 3.23: Accessing auto-mapping
          (A): via the **Mapping** menu, (B): via the **Open EHOOKS
          functions** button, (C): via the context menu in the "Input" and
          "Output" tabs

  – Right-click in a table in the "Input" or "Output" tab and
  select **Map pre-selected ECU Labels (*[13])** from the
  context menu.

Unmapped messages and pre-selected ECU variables (cf.
page 44) with matching names are mapped. Module names in
labels of local messages and record names are not considered
(see also the descriptions on page 33 and page 42).

The results are shown in the "Mapping" field.

To use the Get ECU Labels and Map (*) commands

1. Go to the "Input" or "Output" tab.

2. In the combo box[12], define or select a pattern for auto-mapping.

3. Do one of the following:

   – Select **Mapping > Get ECU Labels and Map (*[14])**.

   – Click on the [icon] **Open EHOOKS functions** button and select **Get ECU Labels and Map (*[??])**.

   – Right-click in a table in the "Input" or "Output" tab and select **Get ECU Labels and Map (*[??])** from the context menu.

   If it is not yet running, EHOOKS is started. Matching ECU variables are selected automatically and mapped to messages with matching names.
   Module names in labels of local messages and record names are not considered (see also the descriptions on page 33 and page 42).

3.2.3   Configuring the Scheduling

To map the processes of your ASCET model to dispatch points on the ECU, you first need to map the processes into a "virtual" task called a *bypass function*, and then associate the bypass function with a dispatch point provided by the ECU. This is done in the "Scheduling" sub-tab of the "EHOOKS" tab.

The "Scheduling" sub-tab is described in section ""Scheduling" Tab" on this page.

3.2.3.1   "Scheduling" Tab

The "Scheduling" tab contains the following GUI elements:

A. "Processes" field

   Lists all modules included in the project. Each module can be expanded to display its processes.

B. **Unused processes only** option

   If activated, only processes not assigned to any bypass function are shown in the "Processes" field.

C. [>>] and [<<] buttons

   These buttons are used to map/unmap processes to bypass functions. At least one process in the "Processes" field and one bypass function in the "Bypass function" field must be selected.
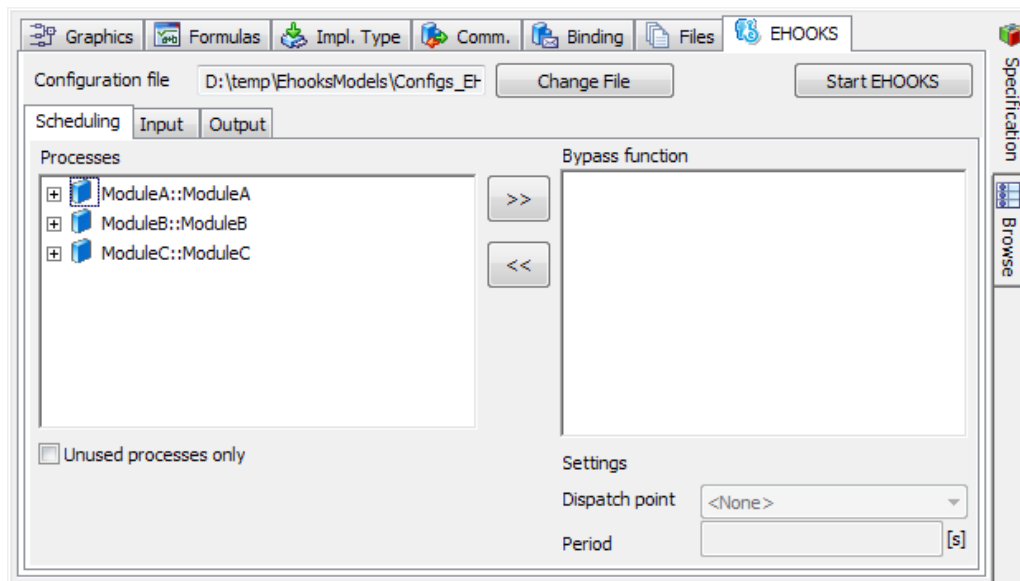
Figure 3.24: "Scheduling" tab

D. "Bypass function" field

Lists all bypass functions in the project. Each bypass function can be expanded to display its assigned processes.

The "Bypass function" field offers a context menu with the following functions:

– **Create from operating system**

Creates bypass functions according to the task list of an existing OS configuration.
See also "To copy an existing OS configuration" on page 62.

– **Add** (<Insert>)

Creates a bypass function.
See also "To create a bypass function" on the next page.

– **Rename** (<F2>)

Renames a bypass function.

– **Delete** (<Delete>)

Deletes a bypass function.

– **Move Up** (<Ctrl> + <↑>) and **Move Down** (<Ctrl> + <↓>)

Moves a process upwards/downwards within the bypass function.

– **Open Module**

Opens a suitable component editor and edits the module that contains the selected process.

– **Remove undefined processes**

Removes undefined processes from the bypass functions.

– **Export**

Opens the "Export Settings" dialog window where you can export mappings to an `*.xml` or `*.csv` file.
See also To export all mappings of one or more tabs on page 66.

– **Import**

Imports mappings from an `*.xml` or `*.csv` file.
See also "To import mappings" on page 67.

– **Create ASCET Link**

Creates an ASCET link that opens the project, and highlights the selected bypass funktion in the "EHOOKS" tab, "Scheduling" sub-tab.

See the ASCET online help for further information on ASCET links.

E. "Settings" field

This field allows to set properties for a selected bypass function.

– "Dispatch point" combo box

Used to associate a bypass function with an ECU dispatch point.
See also "To associate a bypass function with a dispatch point" on the next page.

– "Period" input field

Used to specify a period in seconds. ASCET will use this period for `dT` for all processes mapped to the bypass function.

Possible selections: `<None>`, `<Select>`, previously selected dispatch points

3.2.3.2   Mapping Processes to Dispatch Points

To create a bypass function

1. In the project editor, go to the "EHOOKS" tab and the "Scheduling" sub-tab.

2. In the "Scheduling" sub-tab, right-click in the "Bypass function" field and select **Add** from the context menu.

   A new bypass function is created. Its name is highlighted for editing.

3. Enter a name and press <Return>.

To map a process to a bypass function

1. In the project editor, go to the "EHOOKS" tab and the "Scheduling" sub-tab.

2. In the "Processes" field, select one or more processes (**(1)** in Figure 3.25 on the next page).

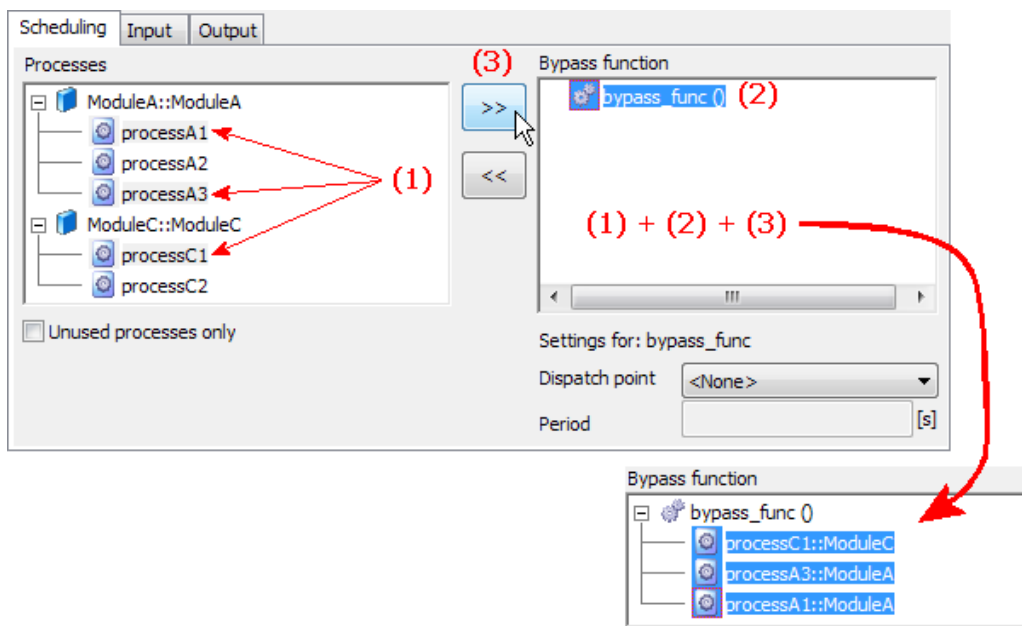3. In the "Bypass function" field, select one or more bypass functions (**(2)** in Figure 3.25 on the next page).

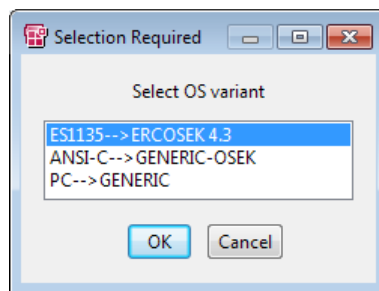Figure 3.25: Mapping processes to bypass functions

4. Click the `>>` button (**(3)** in Figure 3.25 on the next page).

The selected processes are assigned to the bypass function(s).

To copy an existing OS configuration

You can copy a process-to-task mapping from other target/OS combinations (e.g. your PC experiment) to a process-to-bypass function mapping for the EHOOKS Target. To do so, proceed as follows.

1. In the project editor, go to the "EHOOKS" tab and the "Scheduling" sub-tab.

2. In the "Scheduling" sub-tab, right-click in the "Bypass function" field and select **Create from operating system** from the context menu (see Figure 3.26 on the next page).

   The "Selection Required" window opens.

3. Select the combination of target and operating system you want to copy and click **OK**.

   For each task in the copied mapping, a bypass function is created, and the respective processes are assigned.
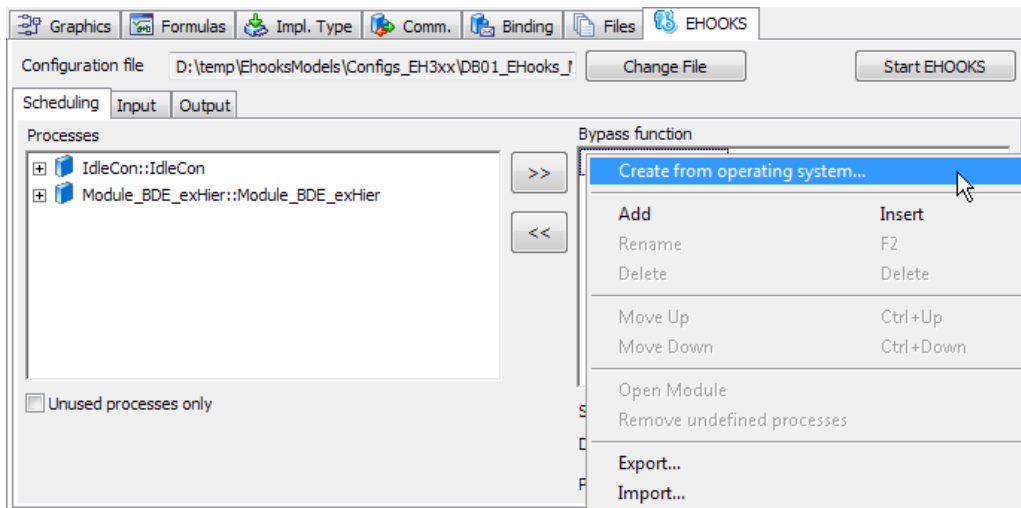

Figure 3.26: Copying an existing configuration from another target

To associate a bypass function with a dispatch point

1. In the project editor, go to the "EHOOKS" tab and the "Scheduling" sub-tab.

2. In the "Scheduling" sub-tab, select a bypass function.

   The "Dispatch Point" combo box is now available.

3. Open the "Dispatch Point" combo box and select `<Select>`.

   If EHOOKS is not already running, it is started now. A
   configuration dialog (see Figure 3.27 on the next page) that lists
   all available dispatch points opens.

4. In the configuration dialog, select the dispatch point you want
   to associate with the bypass function.

5. Close the configuration dialog with **OK**.

   The selected dispatch point is now shown in the "Dispatch
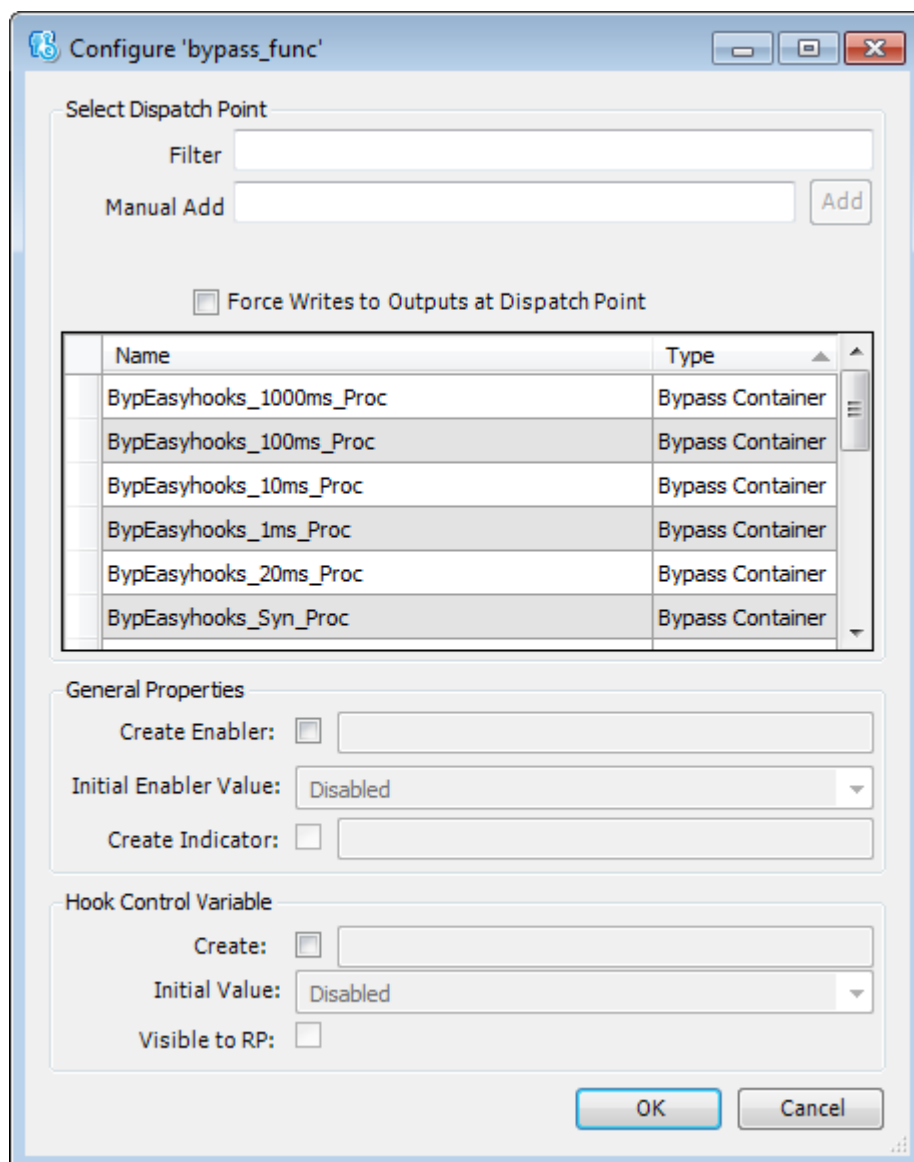   Point" combo box.

Figure 3.27: Selecting a Bypass Container Dispatch Point

To access `dT`

The EHOOKS Target does not currently provide a way to use a `dT` value from the ECU. If your model needs a notion of time, you have to specify a period in seconds that ASCET will use for `dT` for all processes mapped to the bypass function. Proceed as follows.

1. In the "Scheduling" sub-tab, select a bypass function.

2. If necessary, associate a dispatch point (see page 62).

   The "Period" field is now available.

3. In the "Period" field, enter the desired time in seconds.

> **ℹ Note**
>
> If you do not specify a period, ASCET will use an undefined value for `dT`.

3.2.4 Exporting and Importing Mappings

You can export selected message/ECU variable mappings from the "Input" or "Output" sub-tab of the "EHOOKS" tab, or you can export selected mappings of processes to ECU dispatch points from the "Scheduling" sub-tab. Alternatively, you can export all mappings of one to three sub-tabs.

> **ℹ Note**
>
> In this section, the term *mapping* refers to message/ECU variable mapping in the "Input" and "Output" tabs **and** to the mapping of processes to ECU dispatch points in the "Scheduling" tab.

To export selected mappings

1. Go to the tab that contains the mappings you want to export.

2. Do one of the following:
   – In the "Mapping" field of the "Input" or "Output" tab, select one or more mappings.

   – In the "Bypass function" field of the "Scheduling" tab, select one or more bypass functions.

3. Do one of the following:
   – Select **Mapping > Export**.

   – Right-click in the "Mapping" or "Bypass function" field and select **Export** from the context menu.

   If your project contains unsaved mapping changes, you are asked if you want to store the changes.

4. Click **Save** or **Revert** to continue.

   The "Export Selections" dialog window opens. The option **Only Selected Elements in Mapping Table** is preselected.

> ### ℹ Note
>
> You cannot combine the export of selected mappings and the export of all mappings in a tab. If you activate one of the other options, **Only Selected Elements in Mapping Table** is deactivated.

5. Click **OK** to continue.

   A file selection window opens.

6. Select the export format and path and name for the export file.

   You can select either `XML (*.XML)` or `CSV (*.CSV)`.

7. Click **Save** to export the selected mappings.

   The export file is created. When you selected the CSV format, you are informed that the export file is not compatible with ASCET V6.2.0.

8. Read the message carefully, then confirm with **OK**.

To export all mappings of one or more tabs

1. Go to the "Scheduling", "Input" or "Output" tab.

2. Do one of the following:

   – Select **Mapping > Export**.

   – Right-click in the "Mapping" field or in the "Bypass function" field and select **Export** from the context menu.

   If your project contains unsaved mapping changes, you are asked if you want to store the changes.

3. Click **Save** or **Revert** to continue.

   The "Export Selections" dialog window opens. The options in the "Mapping Types" area are preselected.

> ### ℹ Note
>
> You cannot combine the export of all mappings in a tab and the export of selected mappings. If you activate **Only Selected Elements in Mapping Table**, the other options are deactivated.

4. In the "Export Selections" dialog window, select one or more mapping tabs in the "Mapping Types" area.

5. Click **OK** to continue.

   A file selection window opens.

6. Select the export format and path and name for the export file.

   You can select the `XML (*.XML)` or `CSV (*.CSV)` format.

7. Click **Save** to export the selected mappings.

   The export file is created.

You can import mappings from a mapping export file.

To import mappings

1. Go to the "Scheduling", "Input" or "Output" tab.

2. Do one of the following:

   – Select **Mapping > Import**.

   – Right-click in the "Mapping" field or in the "Bypass function" field and select **Import** from the context menu.

   If your project contains unsaved mapping changes, you are asked if you want to store the changes.

3. Click **Save** or **Revert** to continue.

   A file selection window opens. You can filter the display for `XML (*.XML)`, `CSV (*.CSV)` or `AEHK.XML (*.aehk.xml)` [15].

4. Select the mapping export file you want to import.

   > **ℹ️ Note**
   >
   > All mappings in the export file will be imported, you cannot se-
   > lect only a part of the mappings in the export file.
   >
   > There is no check if the imported mappings are valid or invalid.

5. Click **Open** to import the mappings in the selected file.

   The mappings in the file are imported according to the rules given in Table 3.5 on the next page.

6. Check the "Scheduling", "Input" and "Output" tabs and correct invalid mappings.

---

[15] old file format used by ASCET to store mapping information

"Input" tab	• Existing mappings are kept. The respective entries in the export file are ignored. • For each ignored mapping, a warning message is issued in the ASCET monitor window: `Import of mapping ("<message name> -> <ECU variable name>")prevented because it is already existent!`
"Output" tab	• Existing mappings are kept. • If a message in an existing mapping can be mapped to multiple ECU write hooks, the mapping in the export file is added. • If a message in an existing mapping cannot be mapped to multiple ECU write hooks, the mapping in the export file is ignored. • For each ignored mapping, a warning message is issued in the ASCET monitor window: `Import of mapping ("<message name> -> <ECU variable name>")prevented because it is already existent!`
"Scheduling" tab	• Existing bypass functions in the "Bypass functions" field are kept unchanged, even if they are not associated with processes or dispatch points. The respective entries in the export file are ignored. • For each ignored mapping, a warning message is issued in the ASCET monitor window. `Import of bypass function label "<bypass function name>"prevented because it is already available!`

Table 3.5: Rules for import of mappings

3.3   Non-Volatile RAM

EHOOKS supports non-volatile RAM since V3.1. In ASCET, parameters are automatically set to non-volatile; other elements can be placed in the NVRAM memory via the **Non-Volatile** option in the element's properties editor.

To assign the Non-Volatile Attribute to an element

1. Open the component that contains the desired element in a component editor.

2. In the "Outline" tab of the component editor, right-click the element and select **Properties** from the context menu to open the "Properties Editor".

3. In the "Attributes" area, activate the **Non-Volatile** option (see Figure 3.28 on this page).

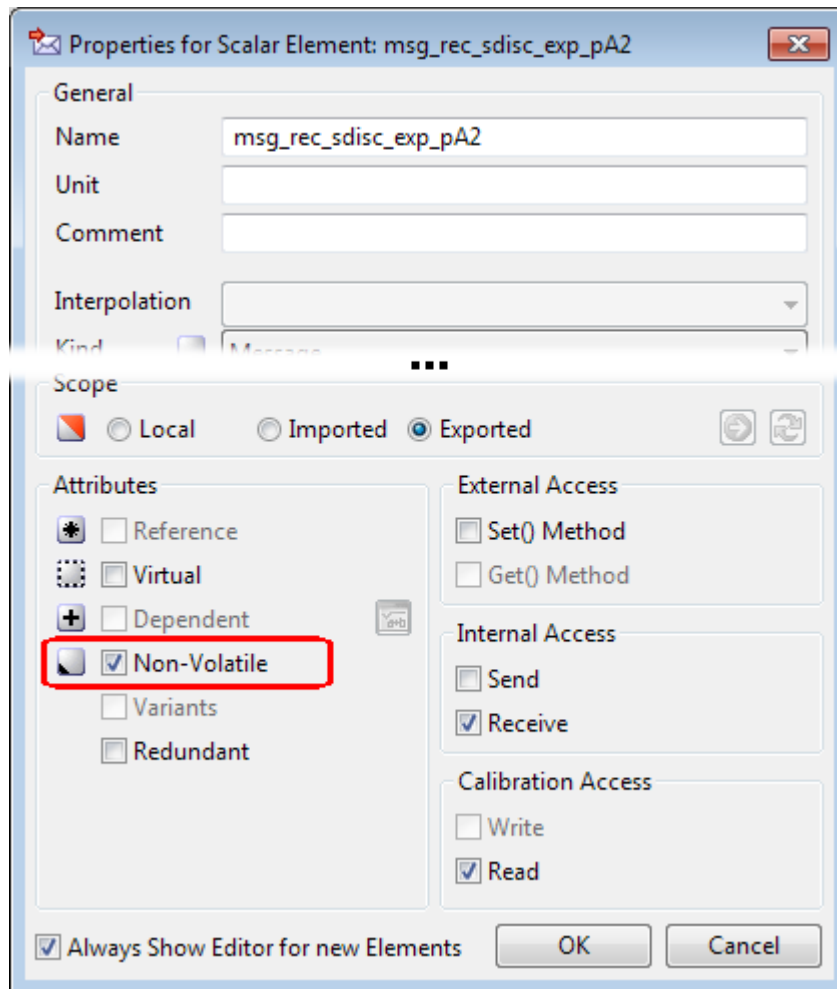4. Click **OK** to close the "Properties Editor".

Figure 3.28: Assigning the Non-Volatile attribute to an element

During the Build process, ASCET generates a `*.six` file in the SCOOP-IX V1.4 format, which contains the **nonVolatile** attribute for `<dataElement>`s; see Listing 3.1 on this page.

```
...
<dataElement>
  ...
    <usage measurement="true"
       virtual="false"
       variant="false"
       nonVolatile="false">
  </usage>
      ...
<dataElement>
```

```
...
```
Listing 3.1: SCOOP-IX V1.4 extract with **nonVolatile** attribute

3.4　Building the ECU Code

To rebuild the ECU hex image, update the ASAM-MCD-2MC file, and generate the SCOOP-IX file [16], select **Build > Build All** or **Build > Rebuild All** from the project editor menu. Alternatively, you can use the keyboard shortcuts: <F7> to build, and <Shift> + <F7> to re-build.

ASCET will generate code for your bypass function(s) and call EHOOKS to rebuild the ECU image and generate a new `*.a2l` file.

The `*.a2l` and `*.hex` files will be located in the places you specified in EHOOKS for output configuration (see section 3.1.4 on page 28).

3.4.1　Using the Code Generator: Object Based Controller Physical

In the "Build" node of the "Project Properties" window, two Code Generators are available: `Object Based Controller Physical` and `Object Based Controller Implementation`, see Figure 3.29 on the next page.

If `Object Based Controller Physical` is selected as code generator, the implementation information in the ASCET Bypass project is ignored, all ASCET *udisc* and *sdisc* data type values are implemented as *uint32* and *sint32*, and the ASCET *log* data type is implemented as *uint8*. The ASCET generated code behaves like the generated code for the physical PC target. When a variable is written to the ECU, the EHOOKS Target will automatically limit the value to the min and max values defined by the ECU.

The min and max values of all Bypass elements in the generated ECU `*.a2l` file are `-9.9995e+36` and `9.9995e+36`.

All *cont* values are implemented as *real32* or *real64* (single or double precision values), depending on the setting of the "Cont Implementation Type" option in the ASCET options, "Targets\EHOOKS\Build" node.

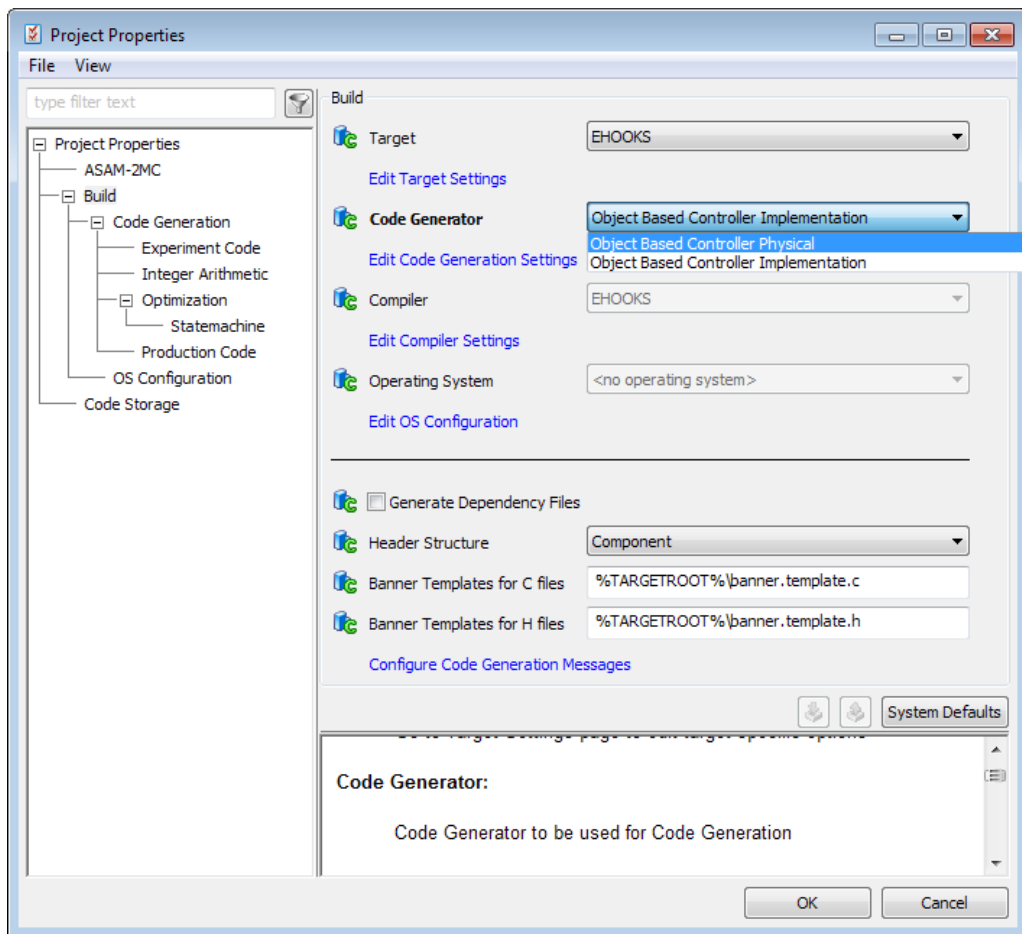[16] all supported EHOOKS versions generate `*.six` files with SCOOP-IX V1.4.

Figure 3.29: Choosing the Code Generator in the project properties, "Build" node

The following settings are available for "Cont Implementation Type" (see also Table 3.3 on page 27):

Phys. Single Precision - Generate all continuous elements as single precision floating point values.

Phys. Double Precision - Generate all continuous elements as double precision floating point values.

> ### ℹ **Note**
>
> This setting applies to **all** projects that use the EHOOKS target.

3.4.2   Generating ECU Code Only

You can generate C code and the `*.ehcfg` file from ASCET by selecting **Build > Generate Code** from the project editor menu or pressing <Ctrl> + <F7>.

> ### ℹ **Note**
>
> ASCET does not update the EHOOKS configuration until the **Generate Code** step is executed. If you have ASCET and EHOOKS open simultaneously, you must perform **Generate Code** to see the ASCET-configured parts of the configuration in EHOOKS.
>
> The SCOOP-IX file is not generated in the **Generate Code** step.

ASCET generates all bypass functions in a single C source file called `asd_bypass_func.c`. This file is located in the directory specified in the ASCET options, "Build\Paths" node, "Code Generation Path" field (see the ASCET online help for details)[17].

Each generated bypass function has the structure shown in Listing 3.2. Instead of direct access to the EHOOKS variable structures, the methods `EH_ARG_PUT_<ECUVar>` and `EH_ARG_SET_<ECUVar>` are generated for the access.

```
EH_USER_BYPASS_FUNC(<function_name>)
{
    /* save the current value of dT for later restoring
       */
    ASD_DT_SCALED_TYPE Saved_ASD_DT_SCALED =
        ASD_DT_SCALED;

  /* Perform RAM initialization. Use default number of
      bytes
```

---

[17] By default, "Code Generation Path" is set to
`<ASCET installation_directory>\CGen`.

```
  ** to initialize. The OTB function will return with
     failure
  ** until the RAM has been initialized.
  */
  if (!EH_InitRam(0)) {
    return 0;
  }


/*-------------------------------------------------------------
 *    Copy the EHOOKS input arguments to ASCET messages
 *-------------------------------------------------------------*/

 /* ... */

 BP_<Message> = ASD_REAL32_TO_IMPL_<Message>(
    EH_IMPL_TO_float_PHYS_<ECUVar>(
    EH_ARG_GET_<ECUVar>(EH_context)));
 ...


/*-------------------------------------------------------------
 *    Copy the EHOOKS output arguments to ASCET
 *    messages for initialization,
 *    if not already done by EHOOKS input arguments
 *-------------------------------------------------------------*/

 /* ... */

   BP_<Message> = ASD_REAL32_TO_IMPL_<Message>(
      EH_IMPL_TO_float_PHYS_<ECUVar>(
      EH_ARG_GET_<ECUVar>(EH_context)));

 /* ASD_CALC_SCALED_DT macro expects the dT value in
    milliseconds */
 ASD_CALC_SCALED_DT(ASD_DT_SCALED, 10U);

 /* *** Execute processes *** */
 <Module>_<Implementation>_<Process>();
 ...


/*-------------------------------------------------------------
 *    Copy ASCET output messages to EHOOKS output
 *    arguments
 *-------------------------------------------------------------*/

 /* ... */
```

```
EH_ARG_PUT_<ECUVar>(EH_float_PHYS_TO_IMPL_<ECUVar>(
    ASD_IMPL_TO_REAL32_<Message>(BP_<Message>)));
...

/* restore the original value of dT */
ASD_DT_SCALED = Saved_ASD_DT_SCALED;

return 1;
}
```

Listing 3.2: Example bypass function structure (EHOOKS-DEV 3.0)

3.4.3   Viewing the ASCET Build Log

ASCET logs code generation and EHOOKS invocation information in the monitor window. Additional information can be found in the file `Makelog.txt` in the same directory as the `asd_bypass_func.c` file (see section 3.4.2 on page 72).

4   Calibrating Bypass Functions

Calibration on bypass functions requires a slightly different approach when using EHOOKS to using other embedded targets.

For non-EHOOKS targets, ASCET itself must generate all data structures, export all measurement and calibration labels to an A2L file and then, after the build stage, extract address information for all symbols from the executable image and patch them into the A2L file.

With the EHOOKS target, ASCET is not in control of the build process; this is managed transparently by EHOOKS-DEV. This means that EHOOKS-DEV is responsible for all data location and the extraction of addresses to generate the updated ECU `*.a2l` file.

The impact of this is that any elements in an ASCET bypass model that need to be available for calibration in the re-built ECU image must be known to EHOOKS. The elements of the following type are supported for calibration:

- scalar elements
- characteristic lines (1D characteristic tables): fixed, normal and group
- characteristic maps (2D characteristic tables): fixed, normal and group
- arrays
- matrices
- records

Restrictions exist that only parameter characteristic tables and distributions are supported for calibration. Variable elements of characteristic tables and distribution types are not shown in the generated ECU `*.a2l` file.

> ### ℹ️ Note
>
> You must ensure that – for parameters – both options in the "Calibration Access" area of the element's properties editor are set, to ensure that EHOOKS adds the elements to the ECU `*.a2l`. Otherwise the build will run successfully, but the elements will be missing.
>
> It is possible to also write the default raster information to the A2L file. To enable this, select the SCOOP-IX version 1.5 in the settings of the EHOOKS target (see also section **??**).
>
> The default raster helps to configure the data acquisition in the calibration tool.

Figure 4.1 on the next page shows how a curve called `MyFixedCurve` is marked for calibration when working with the EHOOKS Target.
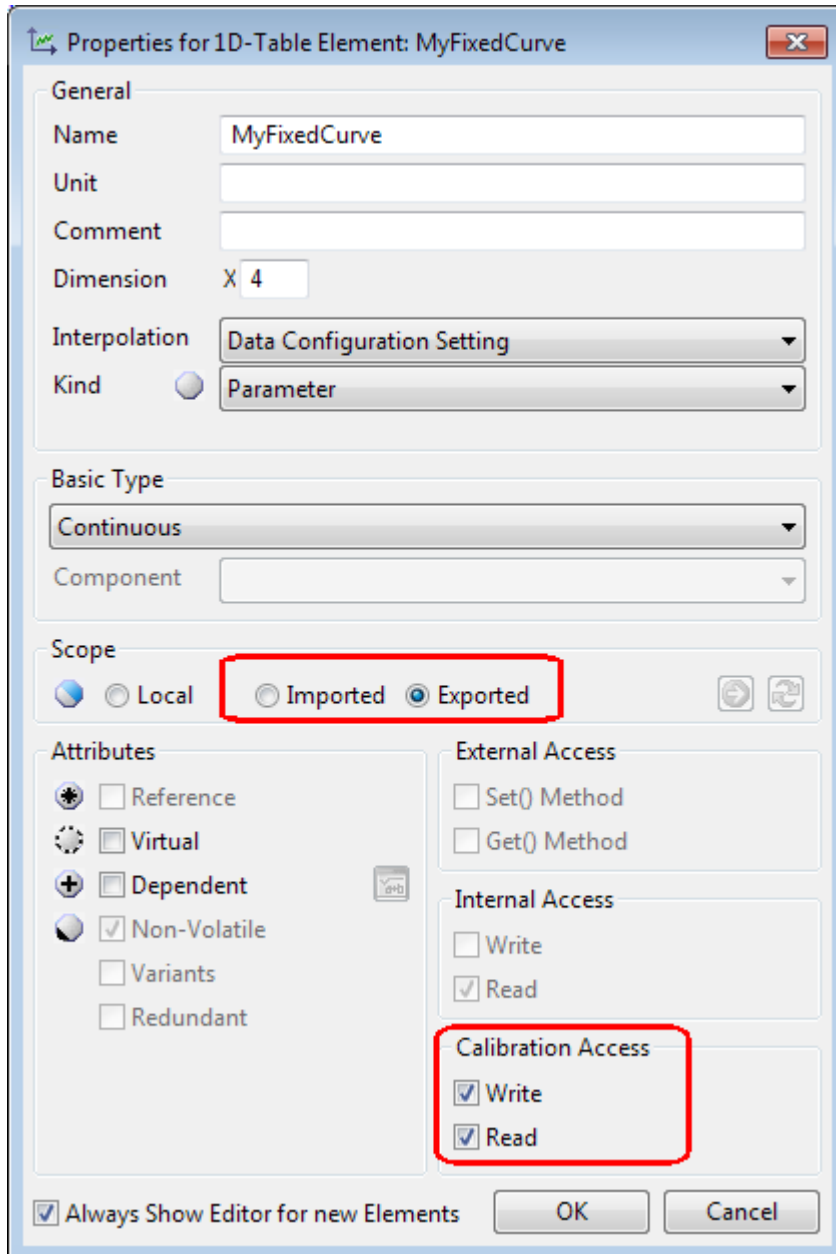
Figure 4.1: Exporting an element

4.1   Calibration of Elements in Classes with Multiple Instances

Calibration of elements in multi-instance components must be explicitly activated. This is done in the ASCET options window, in the nodes of the EHOOKS versions; see Figure 4.2 on this page.
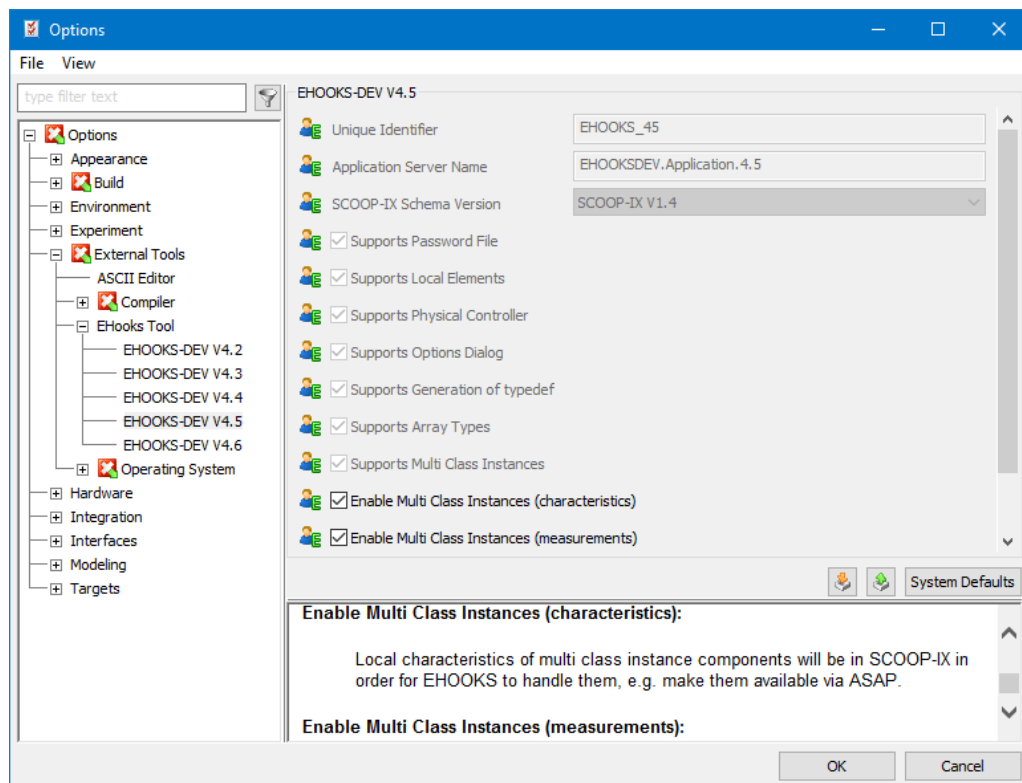
Figure 4.2: Enable the calibration access to characteristics and measurements

> **ⓘ Note**
>
> Calibration access to characteristics in multi-instance classes may consume additional memory, if a calibration method that uses multiple copies is used (e.g. SERAP). If a project runs out of memory, the calibration access can be disabled.

5   Interacting with EHOOKS Control Variables

EHOOKS configurations can define enablers that allow calibration-time and/or run-time control of hooked variables (see Figure 2.2 on page 12).

When an enabler is configured, EHOOKS generates a C variable with the name you specify that acts as a switch to control whether or not the hook is active.

ℹ️ Note

Give your EHOOKS hook control variables names that are valid C names.

For more details on EHOOKS control variables, see the EHOOKS-DEV user guide ([**?**], [**?**], [**?**], [**?**]), sections "EHOOKS-DEV Hook Configuration Properties" and "Configuring Properties of a Variable Hook".

The hook can be enabled and disabled at run-time by writing the following values:

Function	Write Value
Enable	`0x12` (18 in decimal)
Disable	Any other value

You can access this capability from your ASCET bypass function by creating a C code class that writes to the EHOOKS-generated variable. Figure 5.1 shows an example model that disables a hook when a value reaches a specific threshold:
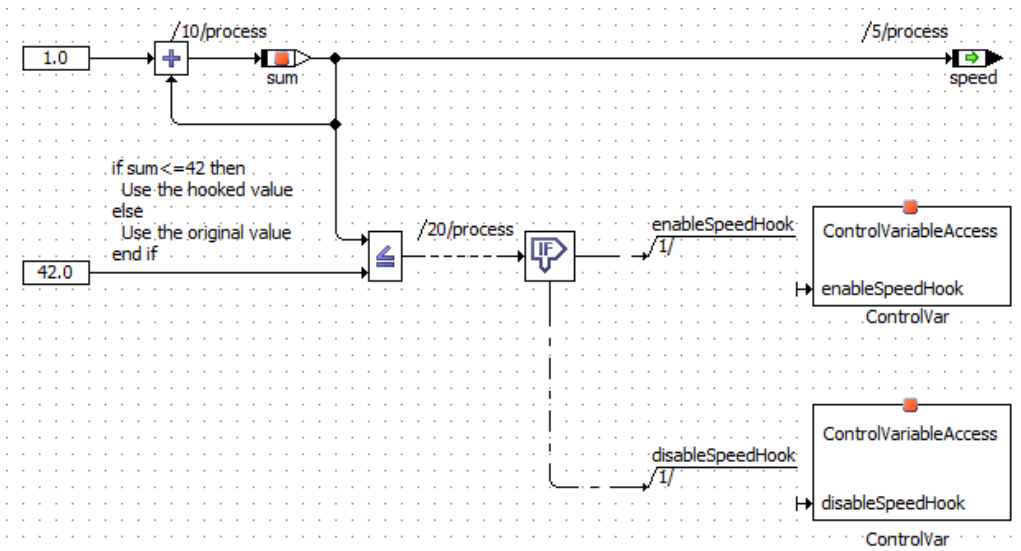
Figure 5.1: Using C code classes to access control variables

To write a C code class to access control variables

You will need to write the C code class(es) to write to the control variables as follows[1]:

1. Create a C code class to store your control methods.

2. Add a method for each variable you need to enable or disable.

   The method can use any valid ASCET method name.

3. At the bottom of the C code editor pane, do the following:

   i. Set "Target" to EHOOKS.

   ii. Set "Arithmetic" to
      Object Based Controller Implementation.

   iii. Leave "Implementation" set to the default.

4. Click on the "Header" tab (in any method or in main – headers are shared across all methods in C code classes) and enter the following code:

   ```
   #include "UserBypassFuncs.h"
   ```

   This header file defines all the available control variables. It is automatically generated by EHOOKS and included in the build process.

5. For each method that must enable the hook, add this code:

   ```
   control_variable_name = 18;
   ```
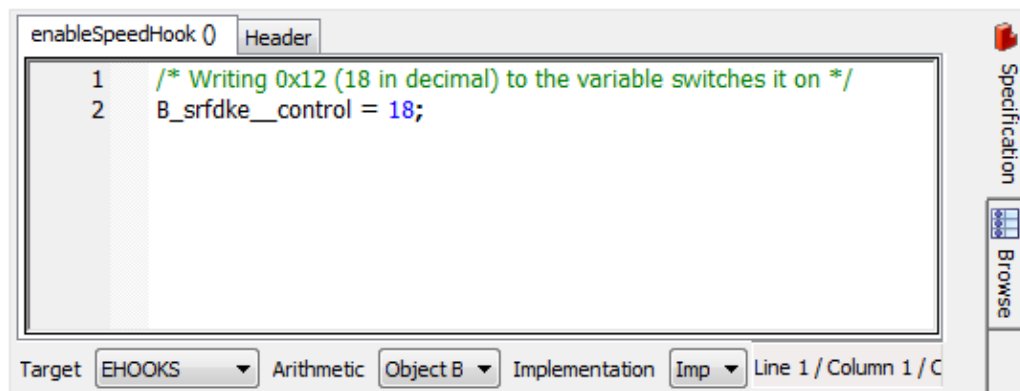
6. If the method must disable the hook, add this code:

   ```
   control_variable_name = 0;
   ```

Figure 5.2 shows a method called enableSpeedHook that writes to a control variable called B_srfdke__control.

The same method for interacting can be used for the Object Based Controller Physical code generator. For the C code classes, you have to select Object Based Controller Physical in the "Arithmetic" combo box (see Figure 5.2 on the next page).

---

[1] There are alternative ways of building this functionality – you are only limited by the capabilities of the C programming language!

Figure 5.2: C code to enable a hook

> **ℹ Note**
>
> The control variable name used by your configured C code class must be identical to the C name of the EHOOKS hook control variable you declare in the EHOOKS configuration.
>
> It is important to remember the following: If a configured hook control variable name is not a valid C identifier, then EHOOKS will automatically convert the name into a valid C identifier by replacing all characters that are not permitted in a C identifier with double underscores (___).
>
> For example, if you call a control variable `MyVariable.control`, then EHOOKS will automatically convert the name to `MyVariable__control`. You must use the converted C name when building C code classes that write to EHOOKS hook control variables.

6 Arithmetic Services and Interpolation Routines

ASCET can interact with user code that is provided outside of ASCET's own code generation process. To do this, ASCET needs to know what code exists and when to use it. This information is provided by `*.ini` files.

During code generation, ASCET uses the information in the `*.ini` files to generate callbacks to user code. At compile time you must provide the implementation of the callbacks you have told ASCET to use. These callbacks are sometimes called *service routines* because they provide services to ASCET.

ASCET uses callbacks in the following cases:

Arithmetic services are used to override the compiler's and/or ASCET's default arithmetic operations. Arithmetic services are *optional* and are disabled by default.

Interpolation routines are used to interpolate between axis points in characteristic lines (1D char tables) or maps (2D char tables). Interpolation routines are *mandatory* if your model uses characteristic lines or maps.

Further information about these topics is provided in the ASCET online help.

The EHOOKS Target handles callbacks using exactly the same mechanisms as all other ASCET embedded targets. This means that the classic use-case, where callbacks are made to access code you provide to the project, works with EHOOKS as well. However, another possibility is available with EHOOKS - using callbacks to access functionality that is already available in the ECU.

> **ℹ Note**
>
> Your ECU supplier must have prepared the ECU to support this use case.

You can also combine both approaches, using callbacks that you provide as C code at build time together with callbacks to services provided by the ECU as shown in Figure 6.1 on the next page.

The following sections explain how to configure your bypass functions for use within the context of an EHOOKS project.
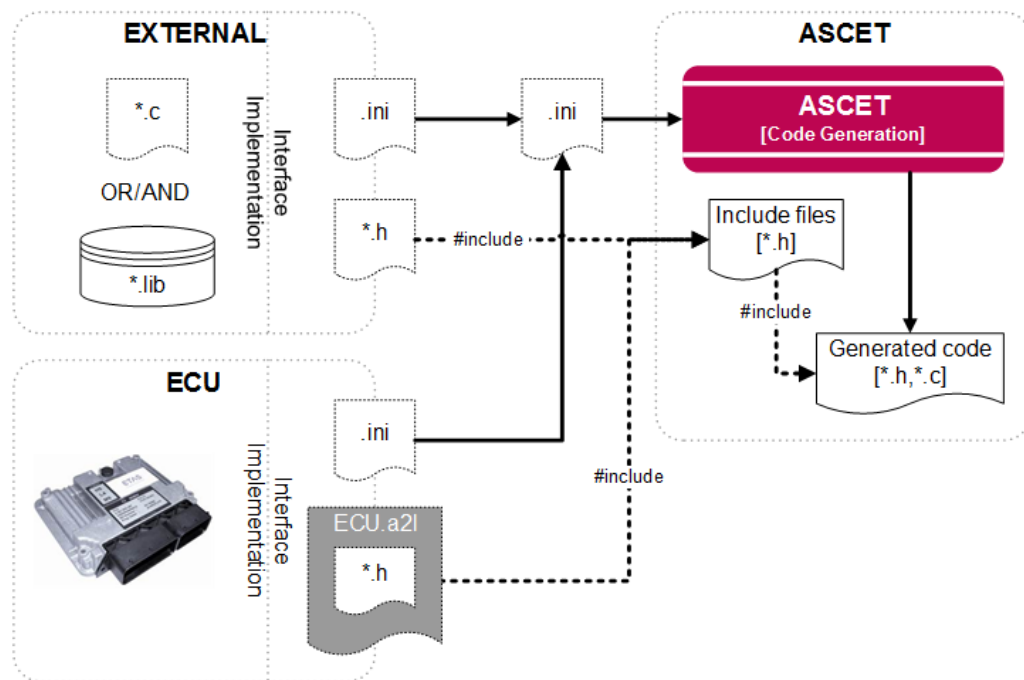
Figure 6.1: Providing callbacks and accessing them in ASCET generated code

6.1   Arithmetic Services

This section provides a brief introduction to principles behind arithmetic services and their use in ASCET. It is not intended to be a comprehensive tutorial; further details are described in the ASCET online help system.

> ### ℹ Note
>
> As for the `Physical Experiment` code generator and the `PC` target, arithmetic services cannot be used for the `Object Based Controller Physical` code generator.

6.1.1   Preparing a Service Set

To use arithmetic services with the EHOOKS Target, you need the following:

- A `services.ini` file that defines which operator signature should be replaced by which arithmetic service routines. This must be located in ASCET's EHOOKS target directory. The default location is `<install_dir>`[1]`\targets\trg_eHooks`.
- The source code and/or libraries for the service routines defined in the `services.ini` file.

The `services.ini` file uses Windows INI file format to define one or more service sets. Each service set appears in a uniquely named section:

---

[1] `<install_dir>` is the ASCET installation directory, e.g., `C:\ETAS\ASCET6.4`

```
[MyServiceSet]
+|*|*|*=Add_NOLIMIT_%t1%%t2%_%tr%(%i1%, %i2%)
-|*|*|*=Sub_NOLIMIT_%t1%%t2%_%tr%(%i1%, %i2%)
*|*|*|*=mul_NOLIMIT_%t1%%t2%_%tr%(%i1%, %i2%)
-|*|*|*=div_NOLIMIT_%t1%%t2%_%tr%(%i1%, %i2%)
...
[MyOtherServiceSet]
+|*|*|*=ADD_%t1%%t2%_%tr%(%i1%, %i2%)
-|*|*|*=SUB_%t1%%t2%_%tr%(%i1%, %i2%)
...
```

Listing 6.1: Service sets in a `services.ini` file

Each un-commented line in the file defines a mapping rule as follows:

`<operator>|<type-signature>=[<return-type>]<function>(<parameters>)`

For example, the rule for + defined in `MyServiceSet` (cf. Listing 6.1 on this page) will cause the replacement of every plus with a call to function `Add_NOLIMIT_%t1%%t2%_%tr%()` where `%t1%` and `%t2%` are the types of the input parameters and `%tr%` is the type of the return value.

When ASCET generates code, each time the operator is required in the context defined by the type signature, a call to the function is generated instead of the normal ASCET code.

For example, Figure 6.2 on the next page shows a model that uses four numerical operations. The inputs are both signed 16-bit integers, and the outputs are signed 32-bit integers. Listing 6.2 on this page shows the code generated by ASCET when no arithmetic service set is selected. Listing 6.3 on the next page shows the code generated by ASCET when the arithmetic service set `MyServiceSet` from Listing 6.1 on this page is selected[2].

```
void SERVICES_IMPL_process (void)
{
    /* process: sequence call #5  */
    out1 = (sint32)(in1 + in2);
    /* process: sequence call #10  */
    out2 = (sint32)(in1 - in2);
    /* process: sequence call #15  */
    out3 = (sint32)(in1 * in2);
    /* process: sequence call #20  */
    out4 = (sint32)(((in2 == (sint16)0) ? in1 : in1 /
        in2));
}
```

Listing 6.2: Code generation without services

---

[2] The code shown has been simplified for clarity. Comments and variable prefixes have been removed.
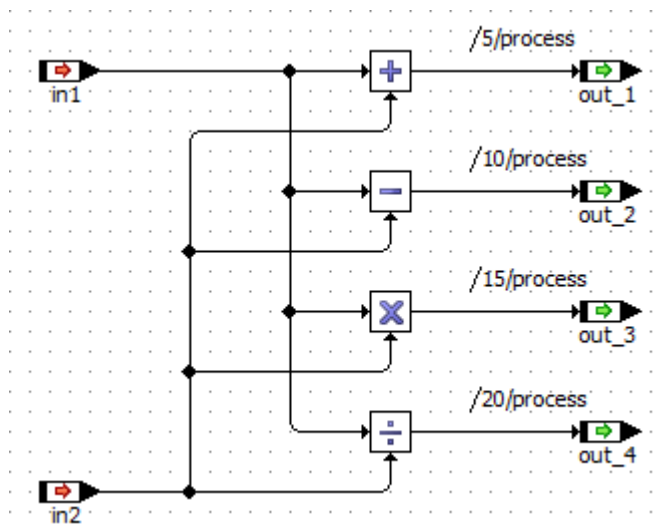
Figure 6.2: ASECT model using arithmetic operators

```c
void SERVICES_IMPL_process (void)
{
    /* process: sequence call #5  */
    out1 = Add_NOLIMIT_s16s16_s32(in1, in2);
    /* process: sequence call #10  */
    out2 = Sub_NOLIMIT_s16s16_s32(in1, in2);
    /* process: sequence call #15  */
    out3 = Mul_NOLIMIT_s16s16_s32(in1, in2);
    /* process: sequence call #20  */
    out4 = (sint32)(((in2 == (sint16)0) ? in1 :
        Div_NOLIMIT_s16s16_s16(in1, in2)));
}
```

Listing 6.3: Code generation with services from `MyServiceSet`

You must provide an implementation for every function referenced by
`services.ini`. The implementation can use any valid C code, including
macro definitions. The following C code examples show the header and
source files that would be required to implement the functions in Listing 6.3
on this page.

```c
#include "a_basdef.h"
sint32 Add_NOLIMIT_s16s16_s32(sint16 x, sint16 y);
sint32 Sub_NOLIMIT_s16s16_s32(sint16 x, sint16 y);
sint32 Mul_NOLIMIT_s16s16_s32(sint16 x, sint16 y);
sint16 Div_NOLIMIT_s16s16_s16(sint16 x, sint16 y);
```

Listing 6.4: Header File: `services.h`

```c
#include "services.h"
uint32 Add_NOLIMIT_s16s16_s32(sint16 x, sint16 y){
    ...
    return ...;
```

```
}
uint32 Sub_NOLIMIT_s16s16_s32(sint16 x, sint16 y{
    ...
    return ...;
};
uint32 Mul_NOLIMIT_s16s16_s32(sint16 x, sint16 y){
    ...
    return ...;
}
uint32 Div_NOLIMIT_s16s16_s16(sint16 x, sint16 y){
    ...
    return ...;
}
```

Listing 6.5: Source File: `services.c`

6.1.2 Using a Service Set

Figure 6.3 on this page shows the interaction between ASCET, EHOOKS and you when integrating arithmetic services.
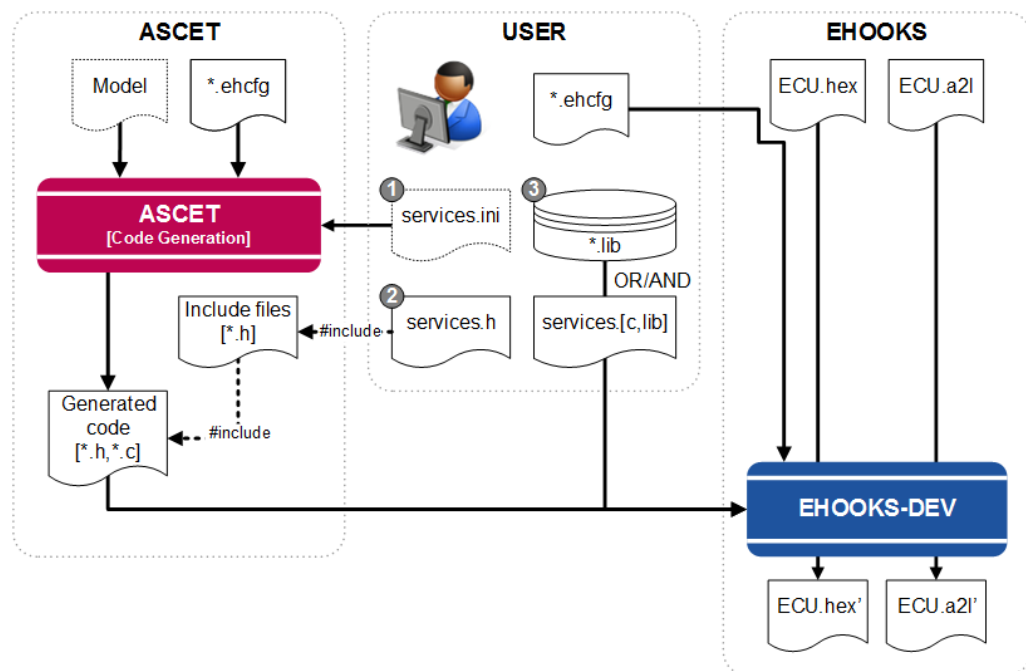
Figure 6.3: Using external arithmetic services with EHOOKS

The following step-by-step guide explains what you need to do.

To use an arithmetic service set:

1. Open the "Project Properties" window and go to the "Build/Code Generation/Integer Arithmetic" node.

2. In the "Arithmetic Service Set" combo box (see also Figure 6.4 on the next page), select the service set you want to use.
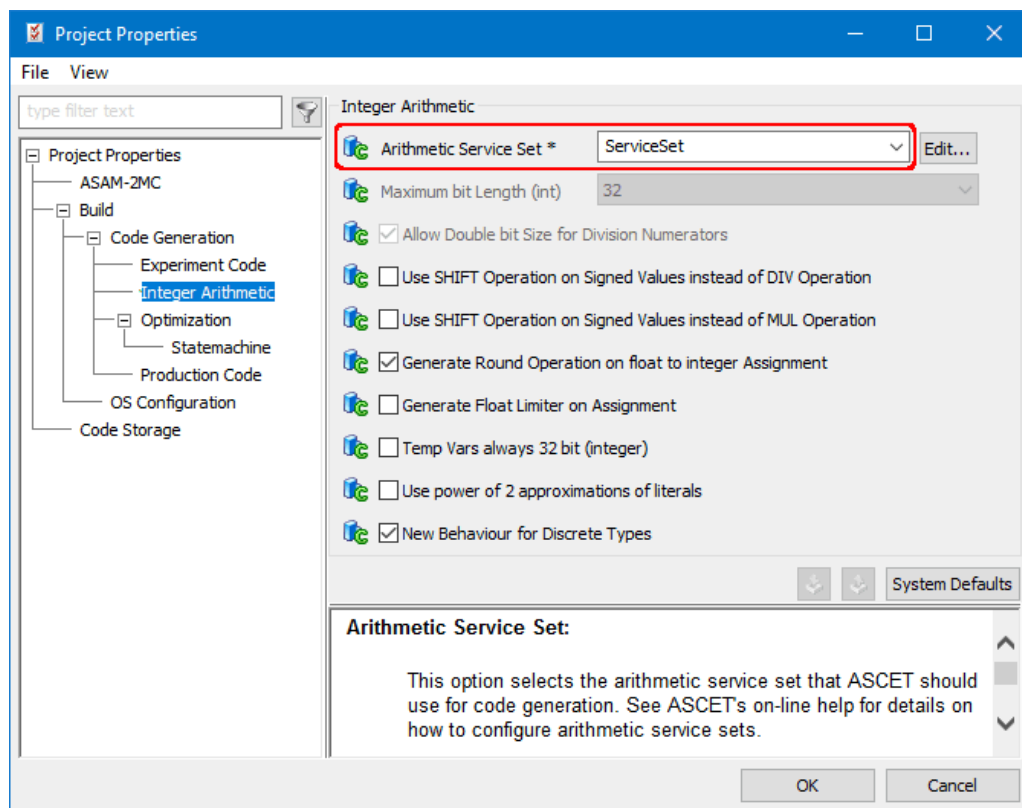
Figure 6.4: Selecting the service set in ASCET

3. Close the "Project Properties" window with **OK**.

4. **#include** the header file(s) for the service implementation in `proj_def.h` in the `targets\trg_eHooks\include` directory.

5. Add the path(s) to the include directories and the names of the source files and/or library files for your arithmetic services to the EHOOKS build as shown in Figure 6.5 on the next page.

When you re-build, ASCET will generate code that includes the calls to your services at the appropriate places using `services.ini`, and EHOOKS will compile and link the service implementations with the ASCET-generated code.

6.2   Interpolation Routines

When your model uses characteristic lines (1D tables) or maps (2D tables), ASCET makes callbacks to C functions called *interpolation routines* to calculate interpolated values.

The following discussion provides some basic information. You can find out more in the ASCET online help: select the **Help > Contents** menu option and – in the help viewer – open the book "Introduction/Basics/Types and Elements/User-defined Interpolation Routines".
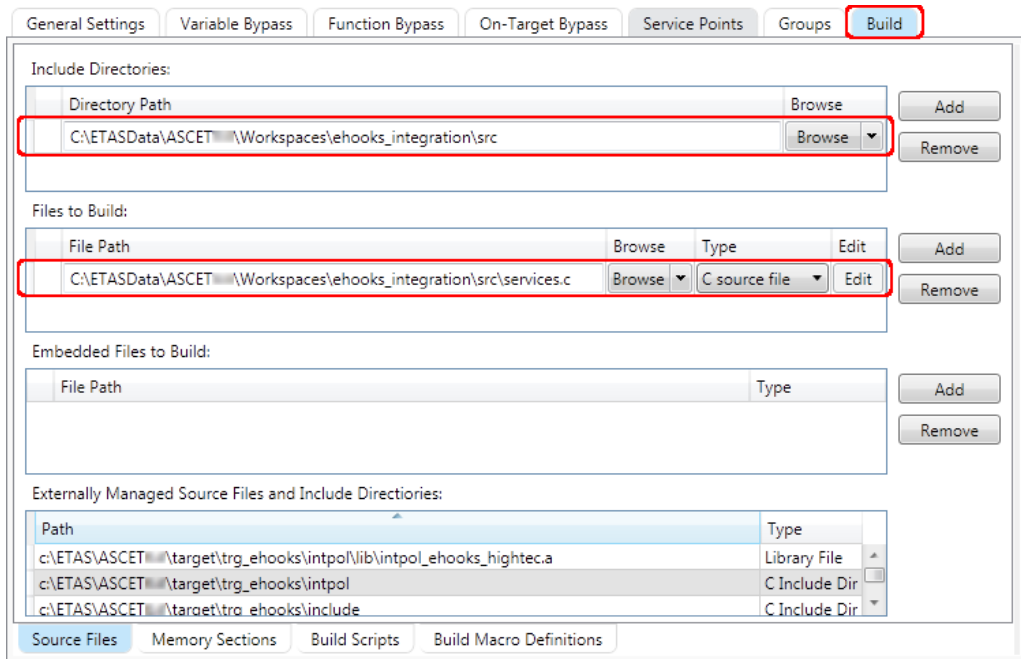
Figure 6.5: Adding a source file to the EHOOKS build

> ### Note
>
> You must use **Help > Contents** to open the entire online help. `<F1>` opens only a part of the online help, and the "Introduction" book may be invisible.

> ### Note
>
> Interpolation routines can also be used with `Object Based Controller Physical` as described in this section.

6.2.1   Understanding Interpolation Routine Use in ASCET

Interpolation routine use in ASCET has both model-specific and target-specific parts. Figure 6.6 on the next page shows how the various parts of ASCET that influence the use of interpolation routines interact in an "out-of-the-box" installation of ASCET.

The following sections explain the contents of these files in more detail.

6.2.1.1   Definition Files

Interpolation definition files tell ASCET what interpolation schemes exist so they can be selected in the model. These definitions allow ASCET to use different interpolation routines for different elements in the same project.

The definitions are located in XML files in `<install_dir>\Tools\Interpolation Routine`. You can see which
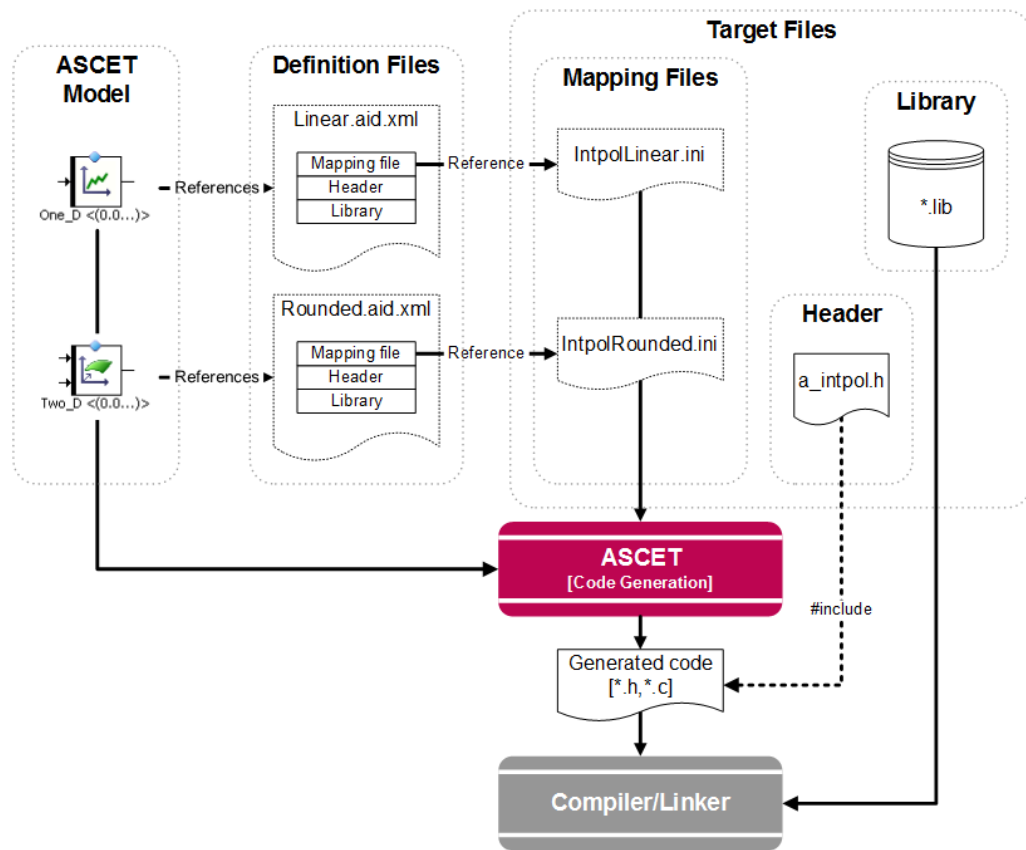
Figure 6.6: From model to code with interpolation routines

definition has been configured for a characteristic line or map by opening the properties editor for the line or map as shown in Figure 6.7 on this page.
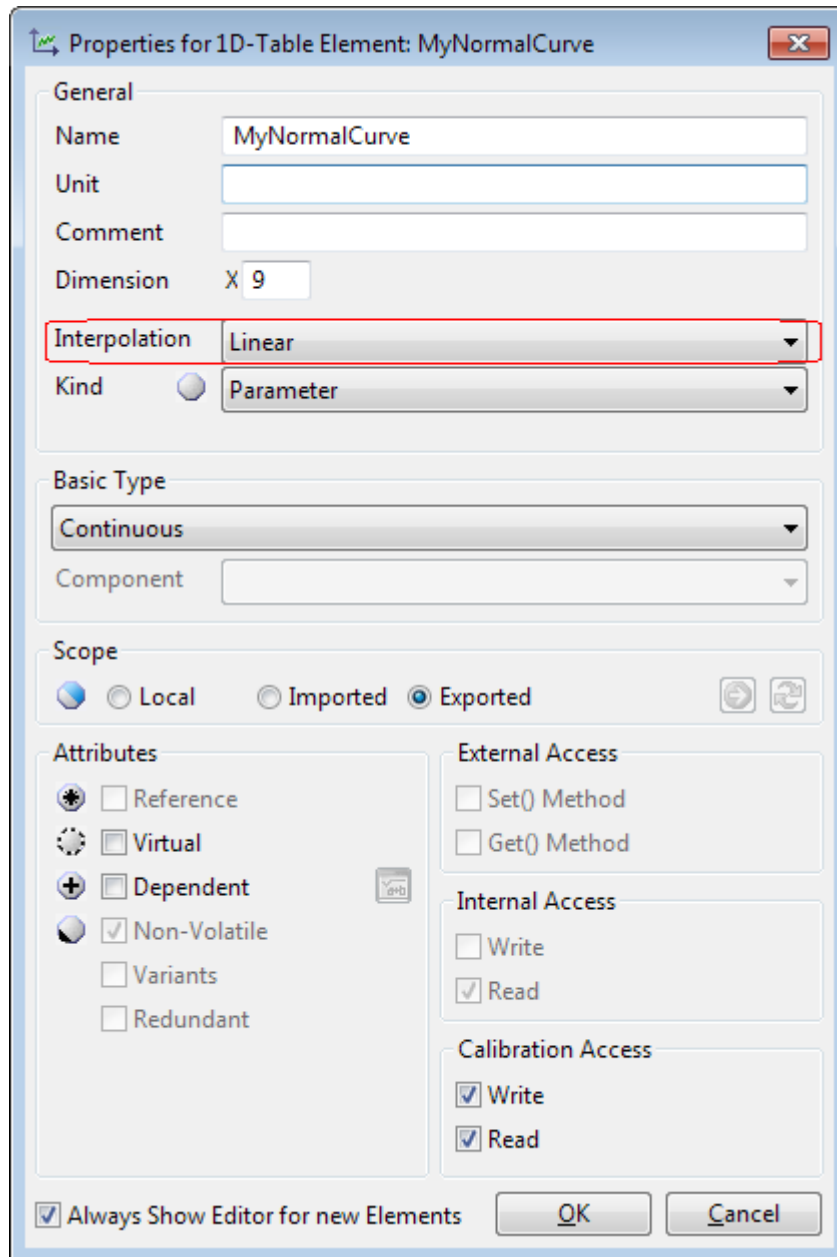
Figure 6.7: Characteristic line using the `Linear` interpolation model

Each interpolation definition specifies the search path to the `*.ini` mapping file that ASCET will use at code generation time. By default, the search path is configured with the following search order:

A. `target\trg_<target>\intpol\Intpol<name>.ini`

B. `target\common\interpolation\Intpol<name>.ini`

This forces ASCET to look in the target directory first and then in the common directory. ASCET will use the first matching definition file it finds.

> ### ℹ️ **Note**
>
> ASCET only installs definitions into the common directory by default.

6.2.1.2 Mapping Files

Mapping files are `*.ini` files, referenced by interpolation definition files, that define the mapping between logical access functions (e.g. `getAt1`, `getAt2` etc.) and the interpolation routines that must be used in the generated code. The file is similar in principle to a `services.ini` file.

There are two sections – `[Experiment]` and `[Production]` –, and each section defines a complete set of mappings. The following example shows the start of the `[Production]` section of the standard `IntpolLinear.ini` file.

```
[Production]
; One D Char Tables
getAt1|*|*=CharTable1_getAt_%tx%%tv%(%ct%,%x%)
getAtFixed1|*|*=CharTableFixed1_getAt_%tx%%tv%(%ct%,%x%)
getX1|*=%ct%->xDist[%i%]
setX1|*=%ct%->xDist[%i%]=%x%
getValue1|*=%ct%->values[%i%]
setValue1|*=%ct%->values[%i%]=%v%
getValueFixed1|*=%ct%->values[%i%]
setValueFixed1|*=%ct%->values[%i%]=%v%
...
```

Note that EHOOKS is classified as a `Production` target as it is an ASCET-SE target.

6.2.1.3 Header Files

Header files declare the interpolation routines and must be included by ASCET code that uses interpolation. Every function named in the `.ini` files must be declared in the header file(s).

6.2.1.4   Library

The interpolation routine library provides the implementation of the interpolation routines declared in the header file(s).

6.2.2   Using the Default Routines

The EHOOKS Target (like all ASCET-SE targets) is supplied with example source code and a pre-compiled library that includes routines for linear and rounded interpolation.

The ASCET EHOOKS Target will automatically use the example routines when generating code. ASCET will add the include file path and the library path to the EHOOKS build for the example library.

> **ℹ Note**
>
> In this release of the EHOOKS Target, the library (`intpol_ehooks_hitec.a`) is built for the Infineon TriCore device using the HighTecs C compiler.
>
> If you need these routines to work with another microcontroller and/or compiler, follow the instructions in `ReadMe_Interpolation.html` in the *`<install_dir>\target\trg_eHooks\intpol`* *directory or contact ETAS for further assistance.*

6.2.3   Using Custom Routines

Using your own interpolation routines in ASCET code generated for the EHOOKS target is possible in the same way as for all ASCET-SE targets. You can either choose to modify the supplied mapping files for linear and rounded interpolation (`IntpolLinear.ini` or `IntpolRounded.ini`) or you can create new definition and mapping files.

6.2.3.1   Modifying an Existing Interpolation Scheme

You can modify an existing interpolation scheme to use your own interpolation routines. The following list explains how:

A. For each interpolation type you use, your interpolation routine mapping file (`IntpolLinear.ini` or `IntpolRounded.ini` as appropriate) needs entries that call your routines. The EHOOKS target is an embedded target, and the entries must be created in the `[Production]` section.

B. Define a C header file that declares every function in your mapping file.

> **ℹ Note**
>
> If you replace the ASCET-supplied header file `a_intpol.h` in `trg_eHooks\intpol`, then ASCET will use the file because it is always *#include*d in `a_basdef.h`).

If you want to use a different file name, you must ensure that it is visible to ASCET-generated code. This can be done by **#include**ing the header file in ASCET's standard location for custom header files: the `proj_def.h` file in `trg_eHooks\include`.

C. Implement your interpolation routines and build a library.

D. Add the include path, source files and/or library files for your interpolation routines to the EHOOKS build as shown in Figure 6.5 on page 87.

6.2.3.2   Creating a New Interpolation Scheme

You can create an entirely new interpolation scheme to use with your models. The following step-by-step guide explains how to create a new set of routines called "CustomInterpolation":

To create a new interpolation scheme:

1. Create a copy of the file `etas.aid.xml.template` in `<install_dir>\Tools\Interpolation Routine`, delete the `.template` suffix and replace `etas` with the name of your interpolation routine definition, e.g. `CustomInterpolation.aid.xml`.

2. Open the file in a text/XML editor of your choice and set the `<DefaultValue>` of the `Identifier` and the `Label` to the name of your interpolation routine.

   The name you specify will be used in the "Interpolation" combo box in the properties editor for a characteristic line or map.

   ```
   <!-- mandatory, fixed options -->
   <OptionDeclaration optionCategory="FIXED"
      xmlCategory=""
      optionClass="EtasStringOption"
      attributeName="Identifier">
      <Group/>
      <Label>Identifier</Label>
      <Description>Unique name for \ASCET
         internal management of this
         interpolation routine.</Description>
      <Tooltip>Identifier for the interpolation
         routine</Tooltip>
   ```

```
        <DefaultValue>CustomInterpolation
            </DefaultValue>
    </OptionDeclaration>
    <!-- required, variable options -->
    <OptionDeclaration optionCategory="FILE"
        xmlCategory=""
        optionClass="EtasStringOption"
        attributeName="Label">
        <Group/>
        <Label>Label</Label>
        <Description>Unique name to display the
            interpolation routine in
            ASCET.</Description>
        <Tooltip>Name of the interpolation
            routine</Tooltip>
        <DefaultValue>CustomInterpolation
            </DefaultValue>
    </OptionDeclaration>
```

> **ℹ️ Note**
>
> It is recommended that the file name and the
> `<DefaultValue>`s of `Identifier` and `Label` are the same
> - this makes it easy for you to identify which description file
> contains which interpolation scheme.

---

3. Save and close the file.

4. Start or re-start ASCET and select **Tools > Options** to open the
   ASCET options dialog window.

5. In the ASCET options dialog window, go to the "Options/Build/
   Interpolation Routine/CustomInterpolation" node and do the
   following:

   i. In the "Interpolation Header" field, enter the header file
      name you want to use.

   ii. In the "Interpolation Library" field, enter the library name
       you want to use.

   iii. Add a "Mapping File" entry, e.g.,
        `%P_TARGET%\CustomInterpolation.ini`.

   iv. If desired, set other options.

   v. Click **OK**.

6. Create the file `CustomInterpolation.ini` in the
   `trg_eHooks` directory.

   For each interpolation type you use, your interpolation routine
   mapping file needs entries that call your routines. The EHOOKS

target is an embedded target, and the entries must be created in the `[Production]` section.

7. Create the C header file with the same name you specified in the "Interpolation Header" and declare every function in your mapping file.

8. Implement your interpolation routines and build a library.

9. Add the include path, source files and/or library files for your interpolation routines to the EHOOKS build as shown in Figure 6.5 on page 87.

To use the new interpolation routine, open the properties editor for your characteristic line or map and select the interpolation routine in the "Interpolation" combo box (see Figure 6.8 on the next page).
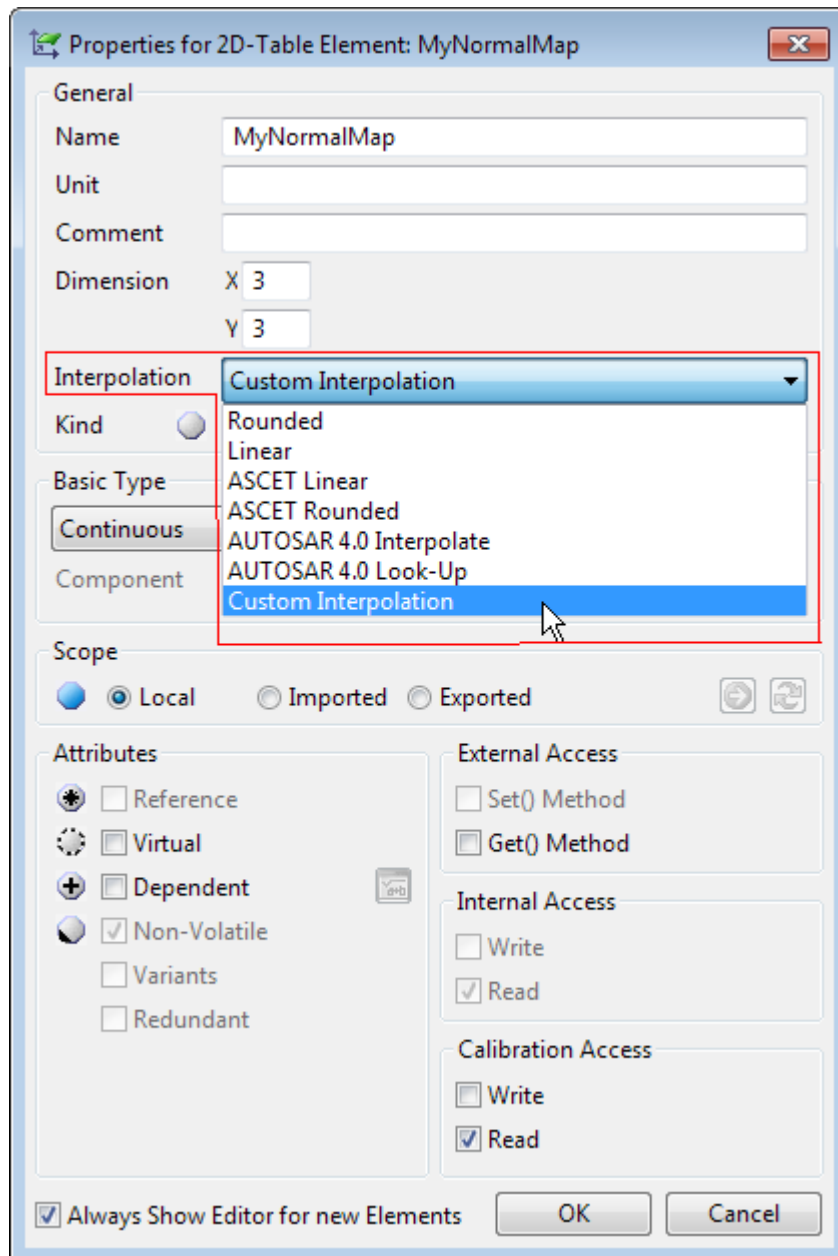
Figure 6.8: Selecting a user-defined interpolation routine

6.3   Callbacks to Existing ECU Code

ASCET can use callbacks to arithmetic services or interpolation routines that are already present in the ECU. Figure 6.9 on this page shows the interaction between ASCET, EHOOKS, the ECU supplier and you when using services provided by the ECU.
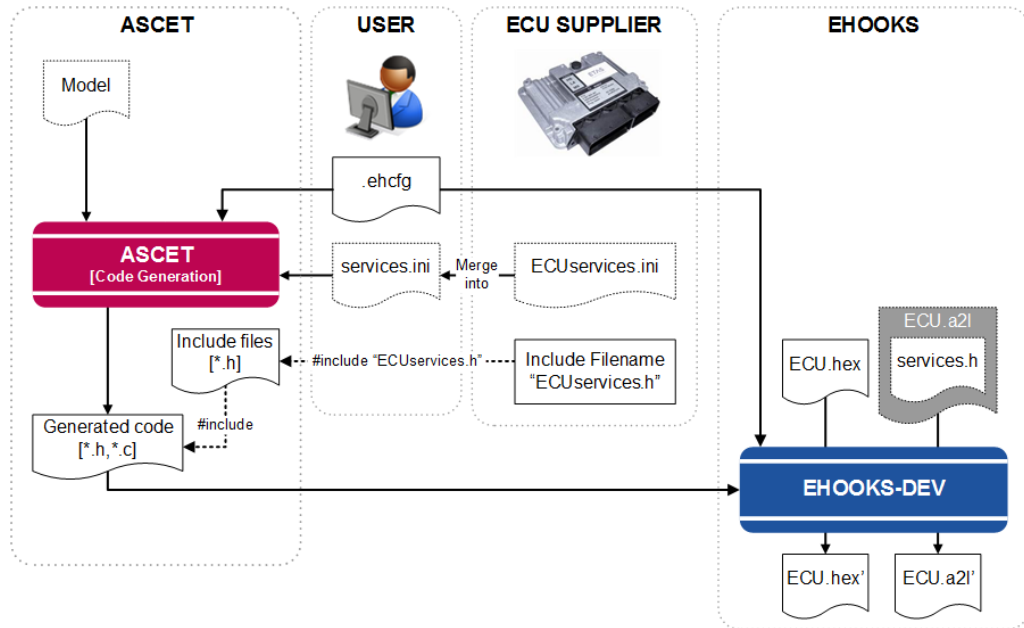
Figure 6.9: Making callbacks to ECU provided functions

Neither ASCET nor EHOOKS can work out what routines are available in the ECU. Your ECU supplier, however, can optionally make routines available when they prepare the ECU using the EHOOKS-PREP tool. They can do this by providing a header file that defines function pointers with the names of the service routines that you can use, and then placing the header file in the ECU_INTERNALS section of the *.a2l file.

For example, Listing 6.6 on this page shows how to create a function pointer to an arithmetic service called Div_limit_s32s32_u16 on the ECU:

```
typedef uint16 (*FPtr_Div_limit_s32s32_u16) (sint32 x,
    sint32 y);
#define Div_limit_s32s32_u16
    ((Fptr_Div_limit_s32s32_u16)(0x1234ABCD))
```

Listing 6.6: Creating a function pointer to an arithmetic service

> ### i Note
>
> It is only possible to make callbacks to services that the ECU supplier makes available to you.

The EHOOKS Target can use callbacks to ECU routines at code generation time instead of (or in combination with) normal externally provided routines. The mechanism is identical to accessing routines that you provide, i.e.:

A. Define `*.ini` files that tell ASCET which routines to use and when.

B. Ensure that the header files for the routines are included by ASCET.

You therefore need the following information to use these routines in ASCET-generated code:

- The `*.ini` file mappings for the ECU routines made available by the ECU supplier.

- The name of the header file(s) that declare the C function names for routines. ASCET needs to **#include** these files [3].

The *only* difference is where the implementation of the routines is found – in the ECU instead of in a C source file or pre-built library.

6.3.1   Arithmetic Services

To use arithmetic services from the ECU

> **ℹ Note**
>
> The casting strategy `Target Optimized` is not permitted when using arithmetic services.

1. Define a service set in the `services.ini` file.

2. Select the service set in the "Project Properties" window, "Build/Code Generation/Integer Arithmetic" node.

3. *#include* the header file(s) for your service implementation in the ASCET `proj_def.h` header file.

When you re-build, ASCET will generate code that includes the calls to the ECU services at the appropriate places, and EHOOKS will compile and link the service implementations with the ASCET-generated code.

---

[3] You only need the name of the file(s) - not the files themselves. Your ECU supplier will embed the header files in the `*.a2l` file when preparing the ECU for EHOOKS. When EHOOKS rebuilds the ECU it will automatically extract the header files and use them in the build process so the ASCET-generated code will compile correctly.

6.3.2   Interpolation Routines

Using interpolation routines from the ECU is very similar to using interpolation routines in other case. You need to do the following:

A. Create or modify (see section 6.2.3 on page 91) an interpolation routine signature mapping file that maps routine signatures to the routines provided by the ECU. For the EHOOKS target, you must create entries in the [Production] section.

B. *#include* the header file(s) for your service implementation in ASCET's proj_def.h header file.

When you re-build, ASCET will generate code that includes the calls to the ECU interpolation routines at the appropriate places.

6.3.3   Mixing Callbacks to Off-ECU and On-ECU Code

You can freely mix callbacks to both Off-ECU and On-ECU code by defining a mapping file (services.ini and/or IntPol*.ini as appropriate) that include mappings to both ECU functions and functions you will provide as external code and/or libraries. You must then include appropriate header files.

It may be the case that you do not know which services will be on the ECU and which will need to exist externally at the point you decide to write your header files. You will have two header files, one for the Off-ECU functions that declares a C function, and one for the On-ECU functions that defines a function pointer for the C function. For example, in the Off-ECU header you might have the following code:

```
uint32 SomeFunction (uint32 x, uint32 y);
```

In the On-ECU header file, you might have a C function pointer to access the same function on the ECU:

```
typedef uint32 (*SomeFunction_Ptr) (uint32 x, uint32
   y);
#define SomeFunction ((SomeFunction_Ptr)(0xABCD1234))
```

This is not a problem if you include the header files for On-ECU functions after those for Off-ECU functions (because this ordering will ensure that the On-ECU functions are used in preference to the Off-ECU functions). For example, assume that there are two header files:

- OFF_ECU_Services.h that declares the function prototypes for Off-ECU functions; and

- ON_ECU_Services.h that includes the function pointer definitions for On-ECU functions.

The correct include file ordering would be:

```
...
#include "OFF_ECU_Services.h"
#include "ON_ECU_Services.h"
...
```

> ### ℹ️ **Note**
>
> The On-ECU services must be included after all header files defining Off-ECU services. The application may not compile if you reverse the include order.
>
> This structure exploits the fact that the On-ECU functions are declared as **#define**s in the `ON_ECU_Services.h` header file. While a function name can appear in both files, it is only a **#define** to a function pointer in the `ON_ECU_Services.h` header file. The C programming language is permissive enough to allow this type of construct, and the C preprocessor will use the last definition, the On-ECU function, in preference to the Off-ECU function.
>
> Most modern C compilers will generate a warning when this occurs, which can be safely ignored.[4]

---

[4] Note, however, that these warnings provide an easy way to check which functions are actually being called on the ECU!

7 Using Libraries

You can use ASCET libraries with the EHOOKS Target in exactly the same way as all other ASCET-SE targets. There are typically two types of library:

Model libraries typically provide ASCET class models that implement common functionality. Model libraries contain pre-defined, complete ASCET models that can be re-used across multiple projects. Each library is self-contained – everything that ASCET needs to generate code is contained in the library itself.

Service libraries also provide ASCET class models, but typically with methods implemented as service routines (**A** in Figure 7.1 on this page) or prototype implementation (**B** in Figure 7.1 on this page). This means that a service library defines only the *interface* between ASCET and some externally provided implementation of the functionality. The library is therefore not self-contained. If you want to use a service library, you will need both the ASCET model *and* the external implementation of the library (either as C source code or a pre-compiled library).
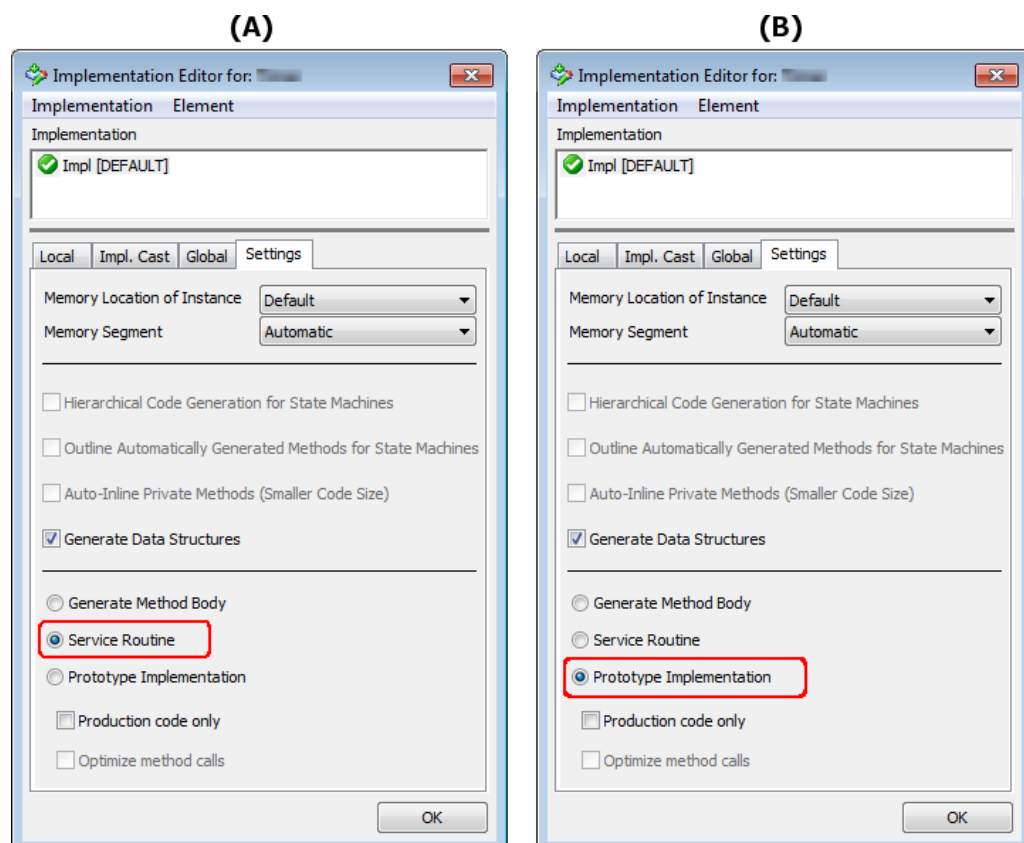
Figure 7.1: Class configured for methods to be implemented as service routines (A) or prototype implementations (B)

> **ℹ️ Note**
>
> To specify service routines and prototype implementations, you must set the "Generate Method Body" option in the EHOOKS target settings to `Use Component Settings`.
>
> Details are given in the ASCET-SE user guide, sections "User-Defined Service Routines" and "Prototype Implementations".

Any given library may contain some classes that are complete models (i.e. those classes are a *model library*) and some other classes that define the interface to services (i.e. those classes are a *service library*).

7.1 Model Libraries

Use of model libraries is straight-forward: import the library into your workspace (or database), and then use the models in your project in exactly the same way you would use models you have built yourself.

ETAS supplies ASCET with two model libraries as standard:

- the `ETAS_SystemLib`
- the `ETAS_MBFS_Library`

These libraries can be found in the `<install_dir>`[1]`\`**export** directory. More information about the functionality of the libraries can be found in the ASCET online help (opened with the **Help > Contents** menu option) and in the `System_Libraries.pdf` document in `...\ETAS\ETASManuals\ASCET<x>.<y>`.

7.2 Service Libraries

When working with service libraries you need the following:

- the ASCET library (as an `*.axl` or `*.exp` [2] export file according to the data model format you are using for your ASCET model)
- one (or more) C header files that define the C interface to the library
- the library itself (either as a pre-compiled library compatible with your ECU or source code)

To use the library, perform the following steps:

A. Import the ASCET library into ASCET using **File > Import**.

B. **#include** the header file(s) for the service library in ASCET's `proj_def.h` header file in the `targets\trg_eHooks\include` directory.

---

[1] `<install_dir>` is the ASCET installation directory, e.g., `C:\ETAS\ASCET6.4`

[2] `*.exp` files can only be imported into ASCET databases. If you use an ASCET workspace, you need an `*.axl` file.

    C. Add the path(s) to the include directories and the names of the source files and/or library files to the EHOOKS build (see Figure 6.5 on page 87).

7.2.1   Controlling Method Names in Generated Code

By default, ASCET generates method names of the following form in the C code[3]:

```
CLASSNAME_IMPLEMENTATIONNAME_MethodName
```

If you mark a method as being implemented as a service, then the service you provide must have this name and use the same signature expected by ASCET.

If the service implementation does not follow the ASCET convention for method names, you can configure ASCET to generate a compatible name by defining a "Symbol" for the method as follows:

To define a symbol for a method

1. Open the component that contains the desired method in the respective component editor.

2. In the "Outline" tab of the component editor, right-click the method and select **Implementation** from the context menu.

   The implementation editor for methods (see Figure 7.2 on this page) opens.

3. Enter the required name in the "Symbol" field, then click **OK**.
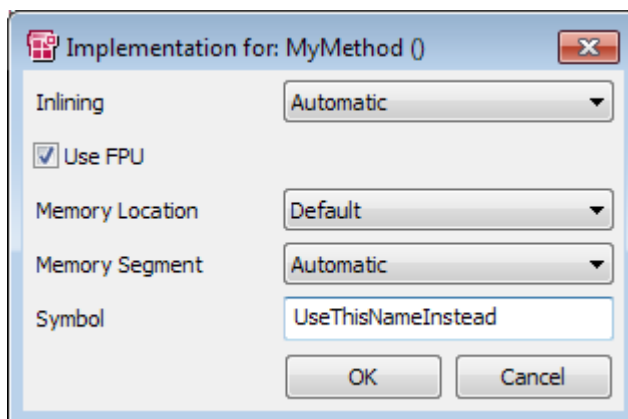
Figure 7.2: Re-defining the symbol name for a method

Figure 7.2 shows how a call to method `MyMethod()` can be modified to be a call to `UseThisNameInstead()`.

------

[3] Note that ASCET capitalizes the module and implementation names by default, so a module with model name `MyClass` will become `MYCLASS`.

If the "Symbol" field is empty, then ASCET generates the default name. If the "Symbol" field is not empty, then ASCET generates a call to the method using the name exactly as written. Names must be valid C identifiers, but they can also use ASCET's template macros (e.g. `%name%`, `%class%`, `%impl%`). For further details, see the ASCET online help.

7.2.2   Optimizing Data Structure Accesses

ASCET-generated component type data structures normally contain a ROM-able part (for constants) and a RAM part (for variables). The ROM part of the data includes a pointer to the RAM part.

If the component only includes variables, then the ROM-able part can be elided in generated code. This removes the ROM structure itself and also removes the data structure pointer indirection, optimizing both time and space. This means that an access of the form:

```
self->RAM_part->element
```

becomes:

```
self->element
```

This optimization is controlled by the option `optimizeCompTypeDescriptor` in `codegen_ehooks.ini`. It is enabled by default in the EHOOKS Target.

7.2.3   Using Services Routines on the ECU

Section 6.3 on page 96 explained how to make callbacks to arithmetic services and interpolation routines that are already present in the ECU hex file.

You can use the same technique to access service routines that already exist on the ECU.

7.3   Working with Formulas

ASCET allows you to define named formulas that specify a fixed-point quantization as shown in Figure 7.3 on the next page. Formulas are defined by *projects*, and module or class implementations can optionally declare that they use a formula with a given name.

The problem with using formulas in this way is that the portability of those implementations can be broken: When the module or class is used in another project, the formula may be undefined. ASCET will report an error at build time for all formulas that are used by modules and/or classes, but are not defined in the project.

Missing formula definitions can be created in the "Formulas" tab of the project editor; see the ASCET online help for details.

Figure 7.3: Formula definitions in a project

The formulas associate with each ECU variable. The decision for automatic conversion of a variable read from the ECU will take the formula specification into account. If the ASCET message and the mapped ECU Label have the same formula specification, no conversion will be generated. This approach has several advantages:

- You do not have to make sure that the formulas used on the ECU and in the bypass model use identical names.

- The accuracy of the value is not changed, since no conversion takes place.

- The runtime for the conversion can be saved.

> **i Note**
>
> You can avoid loss of accuracy even when using the `Object Based Controller Physical` code generator.
>
> If you are using the mixed physical implementation[4], the implementation set does not have to be complete. For each entry in the implementation set, ASCET checks if the formulas of the message and the ECU variable are identical.

---

[4] see the ASCET online help for details

8   Updating Projects from Old EHOOKS-DEV Versions

If you already have an EHOOKS configuration file (`*.ehcfg`) for your project and an old EHOOKS-DEV version (i.e. $\leq$ V4.9), you can just use that file.

To adapt the `*.ehcfg` file to a newer EHOOKS version

1. Open the ASCET options window.

2. In the ASCET options window, go to the "Targets\EHOOKS\Build" node and select a supported EHOOKS version as External Build Tool.

   > **ℹ Note**
   >
   > This setting applies to **all** projects that use the EHOOKS target.

3. Close the ASCET options window.

4. In the "EHOOKS" tab of the project editor, click the **Start EHOOKS** button (see Figure 3.2 on page 19) to open EHOOKS-DEV.

5. If desired, modify your output locations for the `*.hex` and `*.a2l` files in EHOOKS-DEV.

6. Even if nothing needs to be modified, force EHOOKS to save the EHOOKS configuration file (`*.ehcfg`).

   You can do so, e.g., by adding or modifying the contents of a field in the "Project Information" area (marked in red in in Figure 8.1 on the next page).

7. Close EHOOKS-DEV and save the modified EHOOKS configuration file (`*.ehcfg`).

8. In the project editor, select **Build > Build All** or **Build > Rebuild All** to re-build the project with the selected EHOOKS version.

   Alternatively, you can use the keyboard shortcuts <F7> to build and <Shift> + <F7> to rebuild.

If you do *not* have an EHOOKS configuration file (`*.ehcfg`) for your project, proceed as described in chapter 3 "Getting Started with an EHOOKS Project" on page 17.
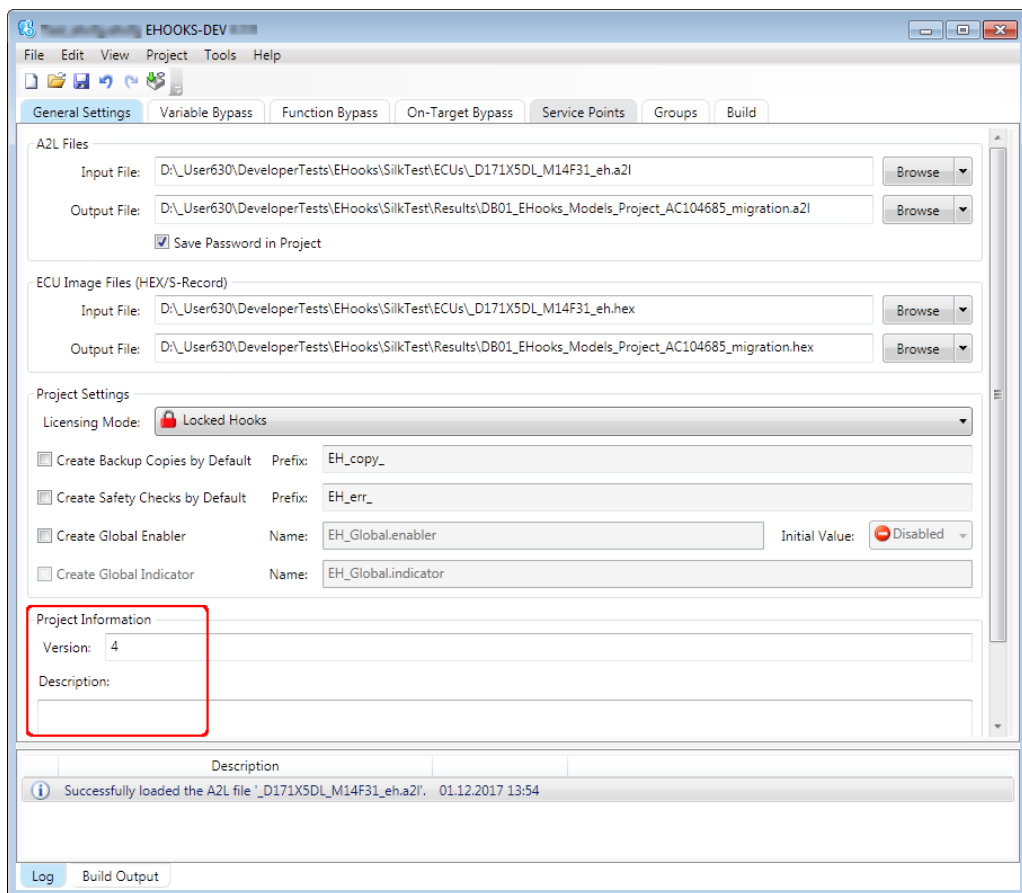
Figure 8.1: Minimal / Start Configuration of an EHOOKS project by providing both a `*.hex` file and an associated `*.a2l` file

9   Contact Information

9.1   Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:
www.etas.com/en/hotlines.php
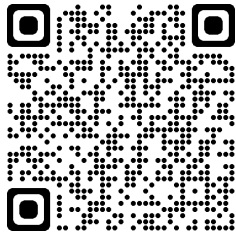
Technical support is available to all ASCET-SE users with a valid support contract. If you do not have a valid support contract, please contact your regional sales office.

You can contact technical support by email or by phone. It is helpful if you can provide technical support with the following information:

- your support contract number
- the version of the ETAS tools you are using
- the version of 3rd-party tools you are using, e.g. compiler tool, version management system etc..
- your `.exp`, `.axl` or `.amd` configuration files
- a description of how to reproduce the error
- the error message you received (if any)

9.2   ETAS Headquarters

ETAS GmbH

Borsigstraße 24	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	Internet:	www.etas.com

List of Figures

List of Tables

Index