

ETAS EHOOKS v5.3  
EHOOKS-DEV

 User Guide

## Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract.

Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

©Copyright 2024 ETAS GmbH, Stuttgart

The names and designations used in this document are trademarks or brands belonging to the respective owners.

ETAS EHOOKS v5.3 - User Guide EHUG R5.3 R01 EN - 03-2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Safety Notice	8
1.2	Privacy Statement	10
1.2.1	Data Processing	10
1.2.2	Technical and organizational measures	10
1.3	Conventions	10
<b>2</b>	<b>Getting Started</b>	<b>11</b>
2.1	Installation	11
2.2	Silent Installation	12
2.3	Licensing	12
2.4	Post Installation Setup and Configuration Steps	12
2.4.1	Installed Versions of EHOOKS Packages	13
2.4.2	EHOOKS-DEV Simulink Configuration	13
<b>3</b>	<b>EHOOKS Overview</b>	<b>16</b>
3.1	EHOOKS Workflow	16
3.1.1	Step 1: EHOOKS Prepared ECU Software	16
3.1.2	Step 2: Creating the Hook Configuration	16
3.1.3	Step 3: Running the EHOOKS-DEV ECU Target Support Tools	17
3.1.4	Step 4: EHOOKS-DEV Generated ECU Software	17
3.1.5	Step 5: Working with the EHOOKS-DEV Generated ECU Software	17
3.2	EHOOKS-DEV Features	18
3.2.1	EHOOKS-DEV Hook Types	18
3.2.2	EHOOKS-DEV Hook Configuration Properties	20
3.2.3	Service Points	27
<b>4</b>	<b>EHOOKS-PREP Dependencies</b>	<b>29</b>
<b>5</b>	<b>Configuring EHOOKS-DEV</b>	<b>33</b>
5.1	General Settings Tab	34
5.1.1	A2L Section	34
5.1.2	ECU Image	36
5.1.3	Project Settings	36
5.1.4	Project Information	37
5.2	Variable Bypass Tab	38
5.2.1	Selecting Variables to Be Hooked	39
5.2.2	Configuring Properties of a Variable Hook	42
5.2.3	Multi-Select Operation	48
5.2.4	Copy and Paste	49
5.3	Function Bypass Tab	50
5.4	On-Target Bypass Tab	52
5.4.1	Configuring Properties of an On-Target Bypass Function	54

5.5	Software Component Bypass Tab	61
5.5.1	Selecting Software Components for Bypass	61
5.5.2	Software Component Bypass Configuration	61
5.6	Service Points Tab	64
5.7	Group Tab	66
5.7.1	Configuring the properties of a Group	67
5.8	Build Tab	69
5.8.1	Configuring Build Source Files	69
5.8.2	Configuring Memory Sections	71
5.8.3	Configuring pre- and post-build scripting	73
5.8.4	Configuring macro definitions	75
5.8.5	Configuring Characteristic Groups	75
5.9	Configuration Consistency Checking, Building and Options	76
5.9.1	Consistency Checking	76
5.9.2	Building Hooked ECU Software	77
5.9.3	Options	77
5.10	Project Actions	78
5.10.1	Convert All Paths	78
5.10.2	Delete all externally configured items	79
5.10.3	On-target bypass	79
5.10.4	Filter files	79
5.10.5	Advanced actions	81
<b>6</b>	<b>Working with Hooked ECU Software in INCA</b>	<b>82</b>
6.1	Run-Time Hook Control and Monitoring	82
6.2	Offset Hooks	85
6.3	Backup Measurement Copies	86
6.4	Safety Checks	86
6.5	Using EHOOKS-CAL and EHOOKS-BYP to Work with Hooks in INCA	86
6.6	A2L Function Groups	89
<b>7</b>	<b>Creating and Working with Simple Internal Bypass</b>	<b>93</b>
<b>8</b>	<b>Creating and Working with External Bypass</b>	<b>95</b>
8.1	Hook based bypass (HBB)	95
8.2	Service based bypass (SBB)	96
<b>9</b>	<b>Creating and Working with On-Target Bypass</b>	<b>97</b>
9.1	Introduction	97
9.2	Step 1: Configure On-Target Bypass Hooks	97
9.3	Step 2: Configure On-Target Bypass Functions	98
9.4	Step 3: Develop the On-Target Bypass Software	98
9.4.1	On-Target Bypass Function Input and Output Parameters	98
9.4.2	On-Target Bypass Function Implementation	99
9.4.3	On-Target Bypass Data Type Conversion	100
9.4.4	Calling ECU functions from On-Target Bypass code	101
9.5	Step 4: Add the On-Target Bypass Files to Configuration	102
9.5.1	Creating a User Definition File	102
9.5.2	Extending the Example to Include a User Definition File	106
9.6	Steps 5: Build and Run the EHOOKS-Created On-Target Bypass Software	108
9.7	EHOOKS On-Target Bypass Global Output Buffer Measurements	108
<b>10</b>	<b>Creating and Working with On-Target Bypass from Simulink</b>	<b>110</b>

10.1	Introduction	.110
10.2	EHOOKS Blocks for Simulink	.110
10.2.1	EHOOKS Configuration Block	.110
10.2.2	EHOOKS ECU Trigger Source Block	.113
10.2.3	EHOOKS ECU Variable Reads Block	.116
10.2.4	EHOOKS ECU Variable Writes Block	.118
10.2.5	EHOOKS ECU Backup Variable Reads Block	.124
10.2.6	EHOOKS ECU Trigger Delegate Block	.126
10.2.7	EHOOKS ECU Hook Control Variable Write Block	.128
10.2.8	EHOOKS ECU Value Parameter Read Block	.129
10.2.9	EHOOKS ECU Variable Read/Write Blocks	.130
10.2.10	EHOOKS ECU Function Call Block	.131
10.2.11	EHOOKS RP-Visible Measurement block	.132
10.2.12	EHOOKS ECU Complex Calibration Parameter Read block	.132
10.2.13	EHOOKS ECU Value Block Cal Param Read block	.135
10.3	Simulink Modelling for On-Target Bypass	.136
10.3.1	Adding the EHOOKS Configuration Block	.136
10.3.2	Adding the EHOOKS Trigger Blocks	.137
10.3.3	Adding the Model and Reading/Writing ECU Variables	.138
10.3.4	Adding the Simulink model	.140
10.4	Building Hooked ECU Software with Simulink On-Target Bypass	.142
10.4.1	Setting the Simulink Configuration Properties	.142
10.4.2	Building the Hooked ECU Software	.143
10.4.3	Running an Experiment with the Hooked ECU Software using INCA	.143
10.5	Advanced Simulink Features	.144
10.5.1	Creating Model Measurements and Calibration Data	.144
10.5.2	Reading Existing Scalar and Complex ECU Calibration Data	.150
10.5.3	Reading from Hooked ECU Variable Backup Copies	.150
10.5.4	Programmatic Control using Control Variables	.151
10.5.5	Communication between On-Target Bypass Functions	.152
10.5.6	Trigger Delegation	.152
10.5.7	Calling an ECU function from within a Simulink model	.154
<b>11</b>	<b>EHOOKS-DEV Reference Guide</b>	<b>156</b>
11.1	EHOOKS-DEV Command Line Usage	.156
11.1.1	Back-End Configuration File	.157
11.1.2	Front-End Configuration File	.157
11.2	EHOOKS-DEV Custom Build Steps	.158
11.2.1	Pre-Generate Scripts	.159
11.3	EHOOKS-DEV Simulink Integration Scripting Interface	.159
11.3.1	Adding EHOOKS Blocks	.160
11.3.2	EHOOKS Simulink APIs	.161
11.4	Special Purpose RAM	.192
11.5	Advanced Project Options	.192
11.5.1	Cached Register Warning Message	.192
11.5.2	Overriding Cached Registers with EHOOKS	.194
11.5.3	Building EHOOKS Code with an Alternative Compiler	.196
11.5.4	Enable Elevated Permissions	.197
11.6	Variable Initialization	.199
11.7	EHOOKS-DEV File Formats	.199
11.7.1	EHOOKS-DEV Project Configuration File	.199
11.7.2	EHOOKS-DEV User Definition Files	.200

11.7.3 EHOOKS-DEV Filter File . . . . .	.200
<b>12 EHOOKS Limitations</b>	<b>201</b>
12.1 Non-Atomic Writes . . . . .	.201
12.2 Timing . . . . .	.201
<b>13 Contact Information</b>	<b>202</b>

# 1 Introduction

This document describes EHOOKS-DEV version 5.3. EHOOKS is used to add hooks and new functionality directly into ECU software with access to only the ECU software and A2L file.

The EHOOKS product tool-chain is made up of the following parts:

- **EHOOKS-DEV**

This is the tool-chain that allows hooks to be inserted into the ECU software. EHOOKS-DEV consists of three parts:

- **EHOOKS-DEV Front-End**

This is the graphical configuration tool that enables quick and easy configuration of EHOOKS projects. The EHOOKS-DEV Front-End is independent of the ECU being used and supports the generation of EHOOKS configurations for all available EHOOKS-DEV ECU Targets.

- **EHOOKS-DEV ECU Target Support**

This is the target-specific tool that processes the EHOOKS project configuration information and inserts the hooks into the ECU software. For each supported ECU there is a separate EHOOKS-DEV ECU Target Support package (please see <http://www.etas.com/ehooks> for the latest information on supported ECUs).

- **EHOOKS-DEV Simulink Integration**

This is the EHOOKS Simulink Blockset, this works in combination with EHOOKS-DEV to enable simple integration of Simulink models with EHOOKS for on-target bypass.

- **EHOOKS-CAL / EHOOKS-BYP**

This is the EHOOKS hook unlocking tool; it uses the INCA API to enable you to work with the hooks placed inside the ECU software by EHOOKS. EHOOK-CAL and EHOOKS-BYP are delivered as a single EHOOKS hook unlocking tool, with the installed license key determining which features are accessible.

---

*NOTICE*

*The need to use the EHOOKS hook unlocking tool depends on the license purchased for EHOOKS-DEV; with certain license keys the EHOOKS hook unlocking tool is not required to work with the ECU software created by EHOOKS. Please contact your local ETAS sales office if you require additional details regarding the supported usage and licensing modes for EHOOKS.*

---

- **EHOOKS-PREP**

This tool is used by the ECU software provider (i.e. the Tier-1) to prepare the ECU software for use with the EHOOKS-DEV tool-chain. This tool is neither necessary nor included with the EHOOKS-DEV package.

This document describes how to install, configure and work with the EHOOKS-DEV, EHOOKS-CAL and EHOOKS-BYP tools. Details of EHOOKS-PREP are provided in a separate EHOOKS-PREP user guide and hence it is not discussed further in this document.

The remainder of this document is structured as follows:

- Section 2 [Getting Started](#) describes how to get started with EHOOKS and how to install, license and configure an EHOOKS setup.
- Section 3 [EHOOKS Overview](#) gives a complete overview of the EHOOKS workflow and the features provided by EHOOKS-DEV.
- Section 3.2.3 [Service Points](#) describes which features and capabilities of EHOOKS can be influenced and controlled by the preparation performed by the ECU software provider using EHOOKS-PREP.
- Section 5 [Configuring EHOOKS-DEV](#) details how to configure EHOOKS using the EHOOKS-DEV configuration tool, describing all aspects of the configuration tool user interface.
- Section 6 [Working with Hooked ECU Software in INCA](#) describes how to work with hooked ECU software created by EHOOKS and illustrates this using INCA. This section also describes how the EHOOKS Hook Unlocker tools (EHOOKS-CAL and EHOOKS-BYP) are used when the licensing mode makes this necessary.
- Section 7 [Creating and Working with Simple Internal Bypass](#) describes how to create simple internal bypass experiments with EHOOKS.
- Section 8 [Creating and Working with External Bypass](#) describes how to create external bypass experiments with EHOOKS.
- Section 9 [Creating and Working with On-Target Bypass](#) describes how to create on-target bypass experiments manually using C code with EHOOKS.
- Section 10 [Creating and Working with On-Target Bypass from Simulink](#) describes how to create on-target bypass experiments using Simulink and the EHOOKS blockset. This section describes in detail how EHOOKS is configured via the EHOOKS blockset for Simulink, how to build the hooked ECU software from within Simulink and how to work with the hooked ECU software containing the Simulink model on the ECU.
- Section 11 [EHOOKS-DEV Reference Guide](#) provides a reference guide for EHOOKS features such as command line options, custom build steps, Simulink integration scripting interface and other advanced features
- Limitations and technical support links are included at the end of the document.

## 1.1 Safety Notice



### **WARNING**

*The use and application of this product can be dangerous. It is critical that you carefully read and follow the instructions and warnings below and in all associated user manuals.*

---

This ETAS product fulfils standard quality management requirements. If requirements of specific safety standards (e.g. IEC 61508, ISO 26262) need to be fulfilled, these requirements must be explicitly defined and ordered by the customer. Before use of the product, customer must verify the compliance with specific safety standards.

This ETAS product enables a user to influence or control the electronic systems in a vehicle or in a test-bench. THE PRODUCT IS SPECIFICALLY DESIGNED FOR THE EXCLUSIVE USE BY



#### PERSONNEL WHO HAVE SPECIAL EXPERIENCE AND TRAINING.

Improper use or unskilled application of this ETAS product may alter the vehicle performance or system performance in a manner that results in death, serious personal injury or property damage.

- Do not use this ETAS product if you do not have the proper experience and training.
- Also, if a product issue develops, ETAS will prepare a Known Issue Report (KIR) and post it on the internet. The report includes information regarding the technical impact and status of the solution. Therefore you must check the KIR applicable to this ETAS product version and follow the relevant instructions prior to operation of the product.

The Known Issue Report (KIR) can be found here: <http://www.etas.com/kir>

- Any data acquired through the use of this ETAS product must be verified for reliability, quality and accuracy prior to use or distribution. This applies both to calibration data and to measurements that are used as a basis for calibration work.
- When using this ETAS product with vehicle systems that influence vehicle behavior and can affect the safe operation of the vehicle, you must ensure that the vehicle can be transitioned to a safe condition if a malfunction or hazardous incident should occur.
- When using this ETAS product with test-bench systems that influence system behavior and can affect the safe operation of the system, you must ensure that the test-bench can be transitioned to a safe condition if a malfunction or hazardous incident should occur.
- All legal requirements, including regulations and statutes regarding motor vehicles and test-benches, must be strictly followed when using this product.
- It is recommended that in-vehicle use of the ETAS product be conducted on enclosed test tracks.
- Use of this ETAS product on a public road should not occur unless the specific calibration and settings have been previously tested and verified as safe.



#### **DANGER**

IF YOU FAIL TO FOLLOW THESE INSTRUCTIONS, THERE MIGHT BE A RISK OF DEATH, SERIOUS INJURY OR PROPERTY DAMAGE

---

THE ETAS GROUP OF COMPANIES AND THEIR REPRESENTATIVES, AGENTS AND AFFILIATED COMPANIES DENY ANY LIABILITY FOR THE FUNCTIONAL IMPAIRMENT OF ETAS PRODUCTS IN TERMS OF FITNESS, PERFORMANCE AND SAFETY IF NON-ETAS SOFTWARE OR MODEL COMPONENTS ARE USED WITH ETAS PRODUCTS OR DEPLOYED TO ACCESS ETAS PRODUCTS. ETAS PROVIDES NO WARRANTY OF MERCHANTABILITY OR FITNESS OF THE ETAS PRODUCTS IF NON-ETAS SOFTWARE OR MODEL COMPONENTS ARE USED WITH ETAS PRODUCTS OR DEPLOYED TO ACCESS ETAS PRODUCTS.

THE ETAS GROUP OF COMPANIES AND THEIR REPRESENTATIVES, AGENTS AND AFFILIATED COMPANIES SHALL NOT BE LIABLE FOR ANY DAMAGE OR INJURY CAUSED BY IMPROPER USE OF THIS PRODUCT. ETAS PROVIDES TRAINING REGARDING THE PROPER USE OF THIS PRODUCT.

## 1.2 Privacy Statement

Your privacy is important to ETAS so we have created the following Privacy Statement that informs you which data are processed in EHOOKS, which data categories EHOOKS 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 or personal-related data.

### 1.2.1 Data Processing

Note that personal or personal-related data respectively data categories are processed when using this product. The purchaser of this product is responsible for the legal conformity of processing the data in accordance with Article 4 No. 7 of the General Data Protection Regulation (GDPR). As the manufacturer, ETAS GmbH is not liable for any mishandling of this data.

When using the ETAS License Manager in combination with user-based licenses, particularly the following personal or personal-related data respectively data categories can be recorded for the purposes of license management:

- Communication data: IP address
- User data: UserID, WindowsUserID

### 1.2.2 Technical and organizational measures

This product does not itself encrypt the personal or personal-related data respectively data categories that it records. Ensure that the data recorded are secured by means of suitable technical or organizational measures in your IT system.

Personal or personal-related data in log files can be deleted by tools in the operating system.

## 1.3 Conventions

The following typographical conventions are used in this document:

---

<code>OCI_CANTxMessage msg0 =</code>	Code snippets are presented in a monospaced font.
Choose <b>File -&gt; Open</b> .	Meaning and usage of each command are explained by means of comments. The comments are enclosed by the usual syntax for comments.
Click <b>OK</b> .	Menu commands are shown in boldface.
Press <Enter>	Buttons are shown in boldface.
The "Open File" dialog box is displayed.	Keyboard commands are shown in angled brackets.
Select the file <code>setup.exe</code>	Names of program windows, dialog boxes, fields etc. are shown in quotation marks.
<i>A distribution</i> is always one-dimensional table of sample points.	Text in drop-down lists on the screen, program code, as well as path- and file names are shown in the Courier font.
	General emphasis and new terms are set in <i>italics</i> .

---

## 2 Getting Started

Working with EHOOKS first requires the relevant parts to be installed and valid license keys obtained. After installing EHOOKS, and depending on the use-case and the configuration of the installation PC, it may be necessary to perform some post-installation setup and configuration steps. Once this has been accomplished EHOOKS can be used.

This section details how to perform all of these tasks.

### 2.1 Installation

EHOOKS-DEV V5.3 provides support for Windows 8.1 and 10 operating systems. EHOOKS-DEV is delivered as three separate packages

- **EHOOKS-DEV ECU Target Support**

This package provides the command-line tools that actually process the EHOOKS project configuration and insert the hooks into the ECU software. This is the minimal installation package necessary to work with EHOOKS-DEV. However, it is typically desirable to work with EHOOKS via the graphical configuration tool and therefore the package below would normally be installed as well. If EHOOKS-DEV is to be used with different ECU projects, then several EHOOKS-DEV ECU Target Support packages will need to be installed – one for each ECU type with which EHOOKS will be used.

- **EHOOKS-DEV Front-End**

This is the graphical configuration tool that enables quick and easy construction of an EHOOKS project configuration. It is possible to use the EHOOKS-DEV Front-End package on a PC which doesn't have any EHOOKS-DEV ECU Target Support packages installed, but then it will only be possible to generate EHOOKS-DEV configuration files – it will not be possible to perform an EHOOKS build to create new ECU software.

- **EHOOKS-DEV Simulink Integration**

This is the EHOOKS Simulink Blockset, which works in combination with EHOOKS-DEV to enable simple integration of Simulink models with EHOOKS for on-target bypass. To work with the EHOOKS-DEV Simulink Integration package you must have both the EHOOKS-DEV ECU Target Support and EHOOKS-DEV Front-End packages installed on the same PC.

The EHOOKS hook unlocking tool (EHOOKS-CAL and EHOOKS-BYP) is delivered as a single package. The need to use and install the EHOOKS hook unlocking tool depends on the license purchased for EHOOKS-DEV.

The installation of each of these packages is carried out in the same basic manner. First, launch the installer for the appropriate package by running **Install XXX.exe** where **XXX** is the component name.

The first step of the installation process is to confirm that the license agreement and safety hints have been read and are acceptable, and then the path to which the installation should occur can be chosen. After this is done clicking the **Install** button will begin the installation process.

To uninstall EHOOKS choose **Control Panel -> Add or Remove Programs** from the Windows **Start** menu, select the desired EHOOKS package and click **Remove**.

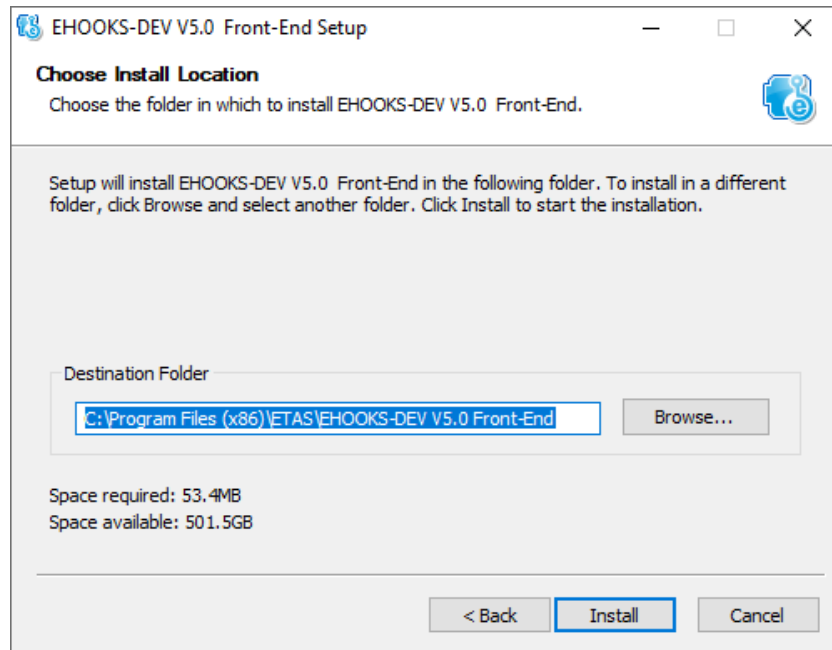


Figure 2.1: EHOOKS Installer

## 2.2 Silent Installation

Each EHOOKS package can also be installed ‘silently,’ that is, without requiring user interaction. This is useful if you wish to create a script or batch file to automate EHOOKS installation.

Pass the /S flag on the command line to enable silent installation.

If you wish to specify a destination directory, use the /D=<path> command-line flag. This must be the last flag on the command line, must be an absolute path, and must not contain quotes, even if the path contains spaces. The installer will use the default path if you do not specify the /D flag.

For example, to install the EHOOKS-DEV Front-End package silently in C:\ETAS, run the following command from an administrator prompt:

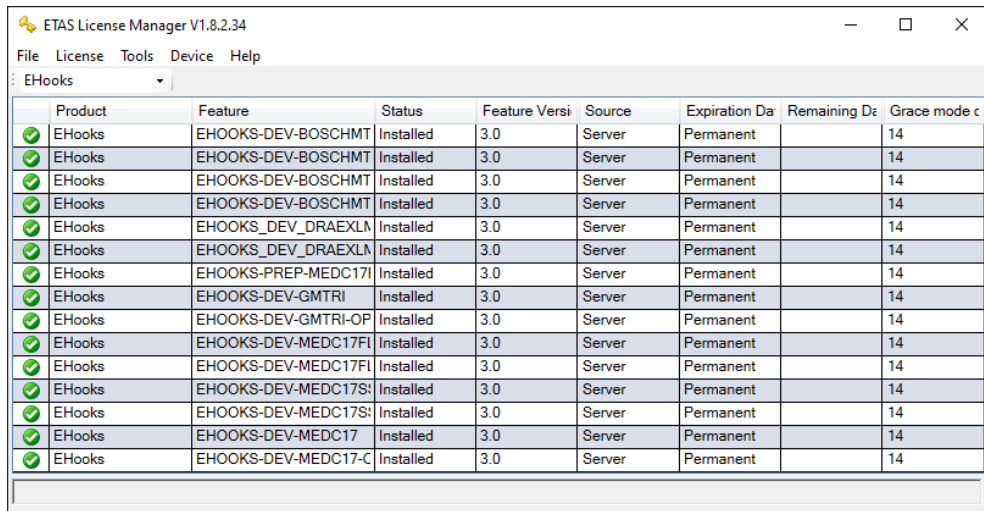
```
"Install EHOOKS-DEV V4.7 Front-End.exe" /S /D=C:\ETAS\EHOOKS-DEV V4.7 Front-End
```

## 2.3 Licensing

EHOOKS V5.3 is licensed using electronic licensing via the ETAS License Manager V1.8.2. To install the license keys you received for EHOOKS launch the ETAS License Manager and select “Add License File” from the file menu. For further details refer to the included documentation and on-line help within the ETAS License Manager.

## 2.4 Post Installation Setup and Configuration Steps

This section describes several setup and configuration steps that are not automatically performed by the EHOOKS-DEV installers and should be carried out manually after the installation process.



Product	Feature	Status	Feature Versi	Source	Expiration Da	Remaining De	Grace mode c
EHooks	EHOOKS-DEV-BOSCHMT	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-BOSCHMT	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-BOSCHMT	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-BOSCHMT	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS_DEV_DRAEXLA	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS_DEV_DRAEXLA	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17I	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-GMTRI	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-GMTRI-OP	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17FI	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17FI	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17S	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17S	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17	Installed	3.0	Server	Permanent		14
EHooks	EHOOKS-DEV-MEDC17-C	Installed	3.0	Server	Permanent		14

Figure 2.2: Managing EHOOKS Licenses with the ETAS License Manager

#### 2.4.1 Installed Versions of EHOOKS Packages

Details of the installed versions of EHOOKS packages can be found via the EHOOKS-DEV menu *Help->About EHOOKS-DEV* launches the About dialog in which all of the installed EHOOKS packages are listed.



Figure 2.3: EHOOKS About Dialog

#### 2.4.2 EHOOKS-DEV Simulink Configuration

To work with the EHOOKS-DEV Simulink Integration Package it is necessary to launch Matlab using a shortcut which will set up your Matlab environment correctly.

This shortcut can be found in the EHOOKS-Dev Front End, in the **Simulink** menu, all compatible versions of Matlab will be visible, for each version you have the option to **Open** or **Create shortcut on desktop**. See figure 2.4

- **Open** launches that version of Matlab with the environment prepared for use with EHOOKS.

- **Create shortcut on desktop** adds the same shortcut to the desktop, so Matlab can be launched with the environment prepared for use with EHOOKS without launching the Front End.

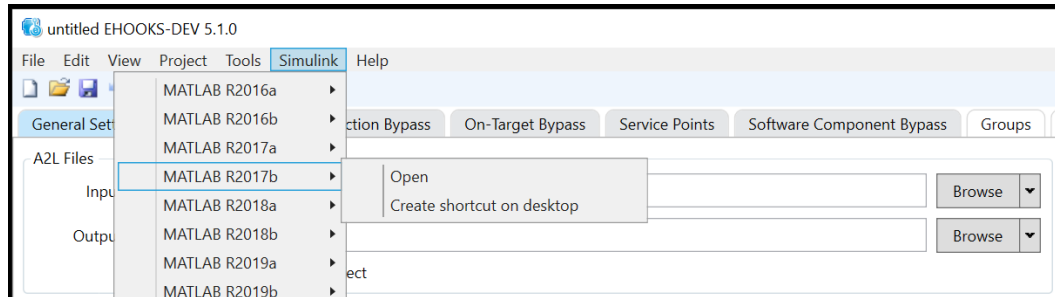


Figure 2.4: EHOOKS-DEV Front End Simulink Shortcuts

---

**NOTICE**

Since EHOOKS version 4.9 it is no longer necessary to run a configuration tool, any libraries required by EHOOKS are now built automatically during the EHOOKS/Simulink build process.

---

#### 2.4.2.1 Manual EHOOKS-DEV Simulink Integration Library Building

The required Simulink libraries can also be built manually. To do this simply execute the batch file from the directory <EHOOKS-DEV ECU Target Installation Directory>\Simulink, as below:

```
Build_libsrc_MATLAB.bat <matlab-version> <target> <matlab-dir>
<matlab-version>: Matlab version number (e.g 9.4)
<target>: EHOOKS ECU target
<matlab-dir>: MATLAB installation directory, e.g. "C:\Program Files\MATLAB\R2018a"
```

#### 2.4.2.2 Manual EHOOKS-DEV Simulink Integration Path Settings

To enable EHOOKS to be used from within Simulink without using one of the EHOOKS generated shortcuts it is necessary to first add the relevant EHOOKS-DEV Simulink Integration package paths to the MATLAB environment. This can be done using the following commands within the MATLAB command prompt:

```
addpath <EHOOKS-DEV Simulink Integration Installation Directory>\MATLAB <ENTER>
add_ehooks_paths <ENTER>
```

To avoid having to do this each and every time MATLAB is loaded, then these same commands can be added to the file <MATLAB Installation Directory>\toolbox\local\startup.m

If the commands are executed correctly you should see the following within the MATLAB command window:

Successfully added the EHOOKS directories to your MATLAB path.

---

**NOTICE**

*If the `startup.m` file does not exist then it can be created using any text editor such as Microsoft Windows Notepad or the file `startupsav.m` can be copied to `startup.m` and used as a template.*

---

### 3 EHOOKS Overview

This chapter gives a brief overview of EHOOKS, including the general workflow and all of the major features. Each of these features will be discussed in greater detail in subsequent chapters of this document.

#### 3.1 EHOOKS Workflow

EHOOKS is used to add hooks and new functionality directly into ECU software with access to only the ECU software and an EHOOKS prepared A2L file.

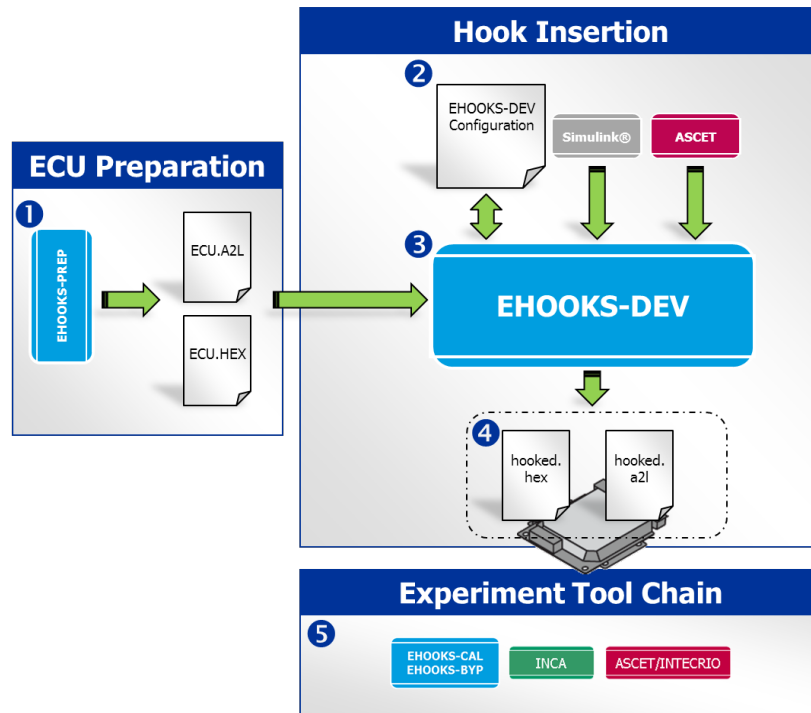


Figure 3.1: Typical EHOOKS Workflow

Figure 3.1 illustrates the typical EHOOKS workflow for adding hooks and new functionality to ECU software and then working with the modified software.

##### 3.1.1 Step 1: EHOOKS Prepared ECU Software

The ECU software provider delivers the ECU software in the usual way – an ECU HEX file<sup>1</sup> and an A2L file. The ECU software will have been prepared for EHOOKS-DEV by the ECU software provider and the A2L file will contain additional information used by EHOOKS-DEV to accurately place the hooks into the ECU HEX file without the need for ECU source code access.

##### 3.1.2 Step 2: Creating the Hook Configuration

EHOOKS-DEV requires the user to describe the hooks and new functionality to be added to the ECU software. This consists of an EHOOKS-DEV configuration and depending on the use-case additional software functionality coming from C source code, ASCET or Simulink®.

Typically the EHOOKS-DEV configuration will be created interactively using the EHOOKS-DEV Front-End, but it is also possible to create this configuration manually as EHOOKS-DEV provides a standardized XML file format (see section [11.7.1 EHOOKS-DEV Project Configuration File](#)). When adding new ECU functionality for on-target bypass, this is typically

<sup>1</sup>EHOOKS supports both Intel HEX files and Motorola S-Record files.



created interactively along with the EHOOKS-DEV configuration using either the ASCET-SE EHOOKS Target (see ASCET-SE EHOOKS Add-on User Guide for full details – this can be found in the ASCET installation directory within the folder <ASCET\_INSTALL\_DIRECTORY>\target\trg\_ehooks\documents) or the EHOOKS-DEV Simulink Integration package (see section [10 Creating and Working with On-Target Bypass from Simulink](#)). Again it is also possible to create new functionality by manually providing C source files (see section [9 Creating and Working with On-Target Bypass](#)).

### 3.1.3 Step 3: Running the EHOOKS-DEV ECU Target Support Tools

EHOOKS-DEV ECU target support tools are then executed to process the ECU's HEX and A2L file along with the EHOOKS-DEV configuration. Typically this is done interactively via the EHOOKS-DEV Front-End, or directly within the ASCET-SE EHOOKS target or the EHOOKS-DEV Simulink integration package.

### 3.1.4 Step 4: EHOOKS-DEV Generated ECU Software

EHOOKS-DEV then analyses and modifies the ECU HEX and A2L file in line with the specified EHOOKS-DEV configuration to generate a new ECU HEX and A2L file containing the requested hooks and new functionality.

### 3.1.5 Step 5: Working with the EHOOKS-DEV Generated ECU Software

The EHOOKS-DEV generated ECU software needs to be flashed into the ECU in the normal way.

---

#### NOTICE

*It is important to ensure that, where necessary, a special purpose calibration data set is used that disables the standard run-time checksums of ECU memory. As EHOOKS-DEV has modified the ECU HEX File these checks will fail and may prevent the ECU from functioning unless they are disabled. Please contact the ECU software provider for details on the appropriate calibration data set to achieve this necessary behaviour.*

---

The A2L file can then be used with the following tools:

- INCA: To measure and calibrate the new ECU software
- INTECRIO: To set up an external bypass experiment consisting of ASCET models, Simulink models and/or C code
- ASCET-RP: To set up an external bypass experiment consisting of ASCET models

To work with the EHOOKS-DEV generated ECU software it is necessary to connect either EHOOKS-CAL or EHOOKS-BYP to the ECU via INCA to unlock the new functionality. If EHOOKS-DEV-OPEN is used then the use of EHOOKS-CAL or EHOOKS-BYP is not necessary.

---

#### NOTICE

*As part of the ECU preparation process, the ECU software supplier can optionally define additional data to be patched at a specified address into the hooked hex/s19 file when it is built by EHOOKS-DEV. This information can be useful to help the EHOOKS-DEV user in identification and management of hooked hex/s19 files. See section [4 EHOOKS-PREP Dependencies](#) for full details. If this capability is required, please ask your ECU software to enable this for you in the ECU preparation stage.*

---

## 3.2 EHOOKS-DEV Features

### 3.2.1 EHOOKS-DEV Hook Types

#### 3.2.1.1 Constant Bypass Hooks

Figure 3.2 illustrates an EHOOKS-DEV constant value hook. This hook type allows run-time control of whether the original ECU calculation or a constant bypass value – in this case 60 – is used for the hooked variable value.

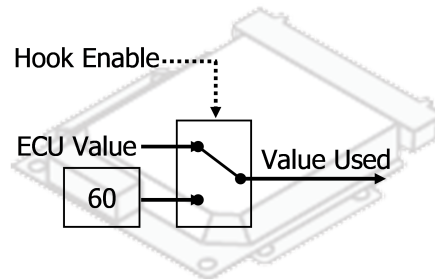


Figure 3.2: Constant Bypass Hook

#### 3.2.1.2 Calibration Value Bypass Hooks

Figure 3.3 illustrates an EHOOKS-DEV calibration hook. This hook type allows run-time control of whether the original ECU calculation or an EHOOKS-DEV-created calibration characteristic is used for the hooked variable value.

In this case INCA can be used to modify the EHOOKS-DEV-created calibration characteristic which in turn directly modifies the hooked ECU variable to a matching value giving the INCA user direct control over the hooked ECU variable's value.

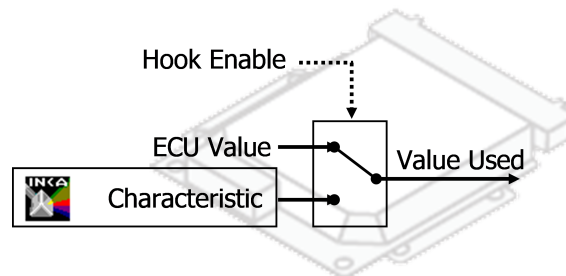


Figure 3.3: Calibration Bypass Hook

#### 3.2.1.3 External Bypass Hooks

Figure 3.4 illustrates an EHOOKS-DEV external bypass hook. This hook type allows run-time control of whether the original ECU calculation or a value calculated on an external rapid prototyping system is used for the hooked variable value.

In this case INTECRIO or ASCET-RP can be used to create a rapid prototype software system that runs on ETAS rapid prototyping hardware (e.g. ES900 series systems). The configured output value from the rapid prototyping software system can be used to modify the hooked ECU variable via the ECU's external bypass connection.

#### 3.2.1.4 On-Target Bypass Hooks

Figure 3.5 illustrates an EHOOKS-DEV on-target bypass hook. This hook type allows run-time control of whether the original ECU calculation or a value calculated by a bypass algorithm

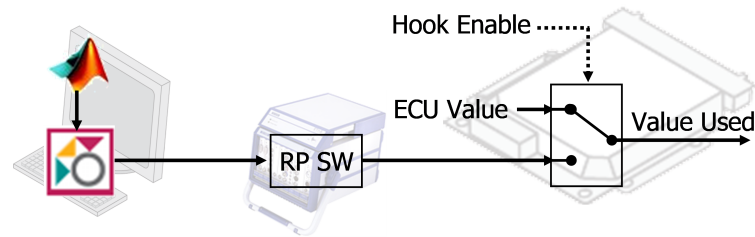


Figure 3.4: External Bypass Hook

running on the ECU is used for the hooked variable value.

On-target bypass allows the integration of a C-code algorithm, ASCET-RP model code or Simulink model code into the ECU software. EHOOKS-DEV allows the on-target bypass to be configured to have access to one or more input variables from the ECU software and to calculate the bypass value for one or more hooked ECU software variables. EHOOKS-DEV also enables the introduction of new measurements and calibration characteristics which the on-target bypass algorithm can then use.

EHOOKS-DEV will automatically compile the provided C-code (or generated C code in the case of an ASCET or Simulink model) and integrate it correctly into the ECU software.

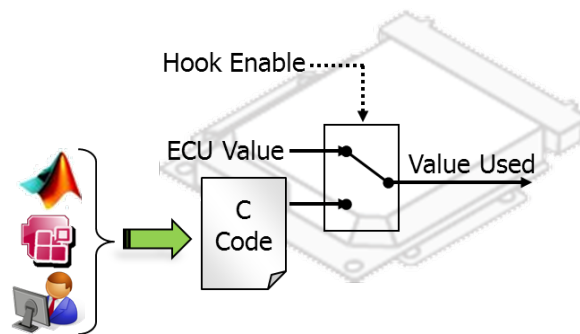


Figure 3.5: On-Target Bypass Hooks

### 3.2.1.5 No-Operation Bypass Hooks

This hook type allows run-time control of whether any value is written to the hooked variable. The use of this hook type will remove any writes to the hooked variable from the ECU software, effectively replacing each write with a NOP instruction.

On-target bypass code can be used to write directly to any variable which has been No-Operation hooked. For this purpose, macros are generated in the UserBypassFuncs.h header file of the form:

```
EH_ARG_DIRECT_PUT_<measurement_name>(<context>, <value>)
```

Where <context> is the context argument of the On-target bypass function.

### 3.2.1.6 Function Bypass Hooks (ECU Process Hooks)

Figure 3.6 illustrates an EHOOKS-DEV function bypass hook.

EHOOKS-DEV allows arbitrary ECU processes to be hooked. A hooked process can then be prevented from executing within the ECU by setting the hook enabler. This functionality will typically be used in conjunction with on-target bypass. In such cases the original ECU process

is replaced with a new implementation (an on-target bypass function) and it may be necessary to disable the execution of the original ECU process to make enough CPU resources available to execute the new implementation.



### WARNING

*Additional caution must be taken when disabling ECU processes; it may be possible to cause serious side-effects by preventing certain ECU processes from executing. This should not be done without ensuring that additional precautions are in place to prevent any failure from causing mechanical damage or personal injury.*

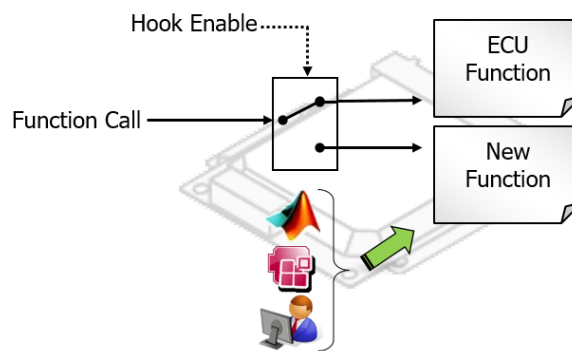


Figure 3.6: Function Bypass Hook

#### 3.2.1.7 Autosar Software Component Bypass Hooks

Autosar Software Components (SWCs) contain any number of “runnables”. Runnables are functions that implement the behaviour of a SWC. EHOOKS-DEV allows permitted SWC, present on the ECU, to be hooked such that a Function Bypass will be created for each runnable belonging to the SWC. Enabled runnables in the original ECU software are replaced with a new implementation provided by the user. Individual enablers allow the user to control at run-time whether each runnable bypass is active.

#### 3.2.2 EHOOKS-DEV Hook Configuration Properties

##### 3.2.2.1 Hook Enablers

Hooks placed into the ECU software by EHOOKS-DEV can be controlled at run-time using calibration characteristics which EHOOKS-DEV creates. These enabler characteristics are regular calibration characteristics whose values can be adjusted at run time using a calibration tool (e.g. INCA). EHOOKS-DEV allows three kinds of enablers to be configured:

1. *Global enabler*

This is a system wide calibration characteristic which, if configured, can be used to enable or disable all of the hooks and changes made to the ECU software by EHOOKS-DEV.

2. *Group enabler*

Groups can be freely created in the configuration and each has its own enabler calibration characteristic. This then allows a group of related hooks to be enabled or disabled altogether with the change of the single group enabler characteristic.

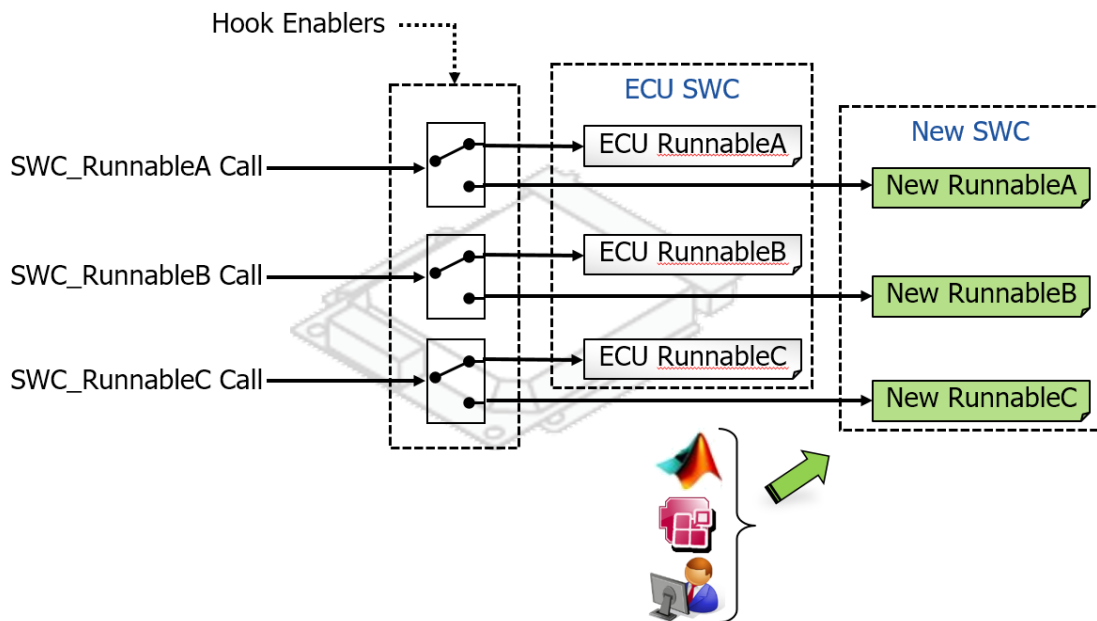


Figure 3.7: Autosar Software Component Bypass Hook

### 3. Hook enabler

Each individual hook can also be configured to have its own enabler calibration characteristic associated. This allows an individual hook to be enabled or disabled independently from the other hooks added to the ECU software by EHOOKS-DEV.

When EHOOKS-DEV creates the new ECU software, by default the initial values of all hook enabler calibration characteristics are set to disabled. This ensures that all changes to the ECU software implemented by EHOOKS-DEV are disabled by default at ECU startup. However, the initial value of the hook enablers can be changed to enabled by the EHOOKS-DEV user, which allows hooks to be turned on at startup.

The logic to control whether a specific hook is enabled or disabled is as follows:

- If a global enabler is configured and is set to disabled, then all hooks are **disabled**.
- If a global enabler is configured and is set to enabled, or a global enabler is not configured; then
  - If any group enabler associated with the hook is enabled, or the hook specific enabler is enabled, or the hook specific control variable (see section 3.2.2.3 [Control Variables](#)) is true, then the hook is **enabled**.
  - If all group enablers associated with the hook are disabled and the hook specific enabler is disabled and the hook specific control variable is disabled, then the hook is **disabled**.

#### 3.2.2.2 Indicators

Indicators are measurement variables that EHOOKS-DEV can place into the ECU software to shadow the value of hook enablers. When the hook code added to the ECU software by EHOOKS-DEV evaluates the status of a hook enabler, the associated indicator measurement

variable (if configured) is updated to match this status.

Indicator measurement variables can be measured at run time using a measurement tool (e.g. INCA) and give feedback on whether the associated hook is enabled or disabled.

---

**NOTICE**

*Indicators can be a very useful diagnostic aid when working with EHOOKS. Sometimes it can appear that a hook is not functioning as expected. In many cases, this can be down to the hook code added by EHOOKS-DEV not actually executing on the ECU, resulting in the bypass value not being written into the hooked ECU variable. This can be due to certain rasters within the ECU not executing (this is especially likely if the ECU is being run in a reduced hardware environment) or due to conditional code within the ECU software structure.*

*If an indicator variable is not updated (i.e. changes between true and false) in synchronization with changes to the associated hook enabler, then the most likely cause is that the hook code is not actually being executed. In such situations the EHOOKS feature of forced-writes (see section 3.2.2.6 [Forced-Write Mechanisms](#)) can still allow the ECU variable to be successfully hooked and updated with the desired bypass value.*

---

When indicator measurement variables are configured, additional hook diagnostic counter measurement variables are automatically created. These provide a deeper insight into the execution of the EHOOKS hook code see section 6.1 [Run-Time Hook Control and Monitoring](#) for more details.

### 3.2.2.3 Control Variables

In addition to hook enablers, EHOOKS-DEV supports control variables to control at run-time whether a specific hook is enabled or disabled. While hook enablers are calibration characteristics which must be controlled via a calibration tool, control variables are regular ECU variables and therefore can be controlled programmatically by on-target or external bypass code. This means that new software logic can be introduced into the ECU via the bypass code which in turn can determine whether specific variable hooks should be enabled or disabled based on certain ECU conditions.

---

**NOTICE**

*To set a control variable to true, i.e. to indicate that the associated hook should be enabled, the control variable should be assigned a value of 0x12.*

*To set a control variable to false, i.e. to indicate that the associated hook should be disabled, the control variable should be assigned a value of 0x0 (although any value other than 0x12 is acceptable).*

*The reason for these special values is to overcome the fact EHOOKS-DEV cannot depend on the ECU to initialize variables it creates within the ECU software when the ECU is powered-on or reset.*

---

The interaction of control variables with hook enablers is described in [Hook Enablers](#).

**NOTICE**

When setting the value from external bypass, the value of the control variable will not be updated in the ECU memory. The control variable is retried from the bypass buffer and is not written back to the control variable for data consistency reasons.

## 3.2.2.4 Replacement, Offset and Multiply Hooks

EHOOKS-DEV supports replacement bypass, offset bypass and multiply bypass. Figure 3.8 illustrates a replacement hook; in this configuration the EHOOKS-DEV inserted hook completely replaces the ECU value with the bypass value.

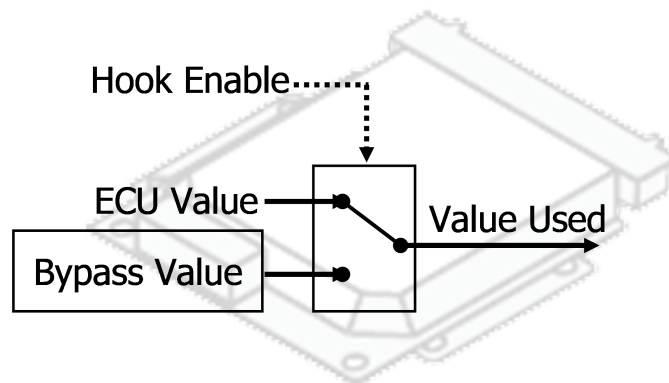


Figure 3.8: Replacement Hooks

Figure 3.9 illustrates an offset hook; in this configuration the EHOOKS-DEV inserted hook is used as an offset to the original ECU value. An EHOOKS-DEV control characteristic is introduced to allow run-time control over whether the value is added to or subtracted from the original ECU value.

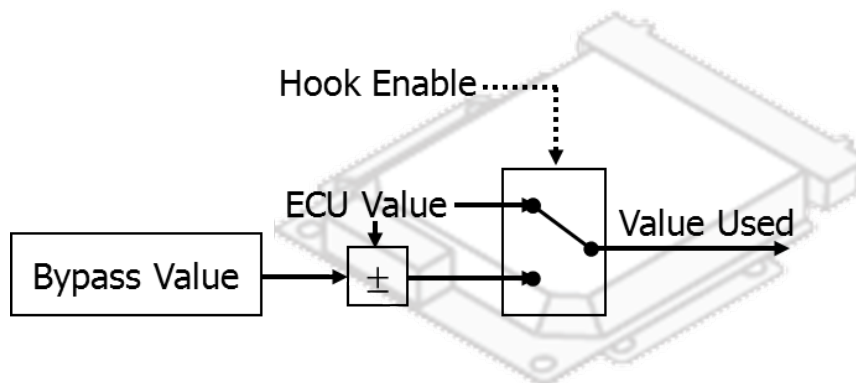


Figure 3.9: Offset Hooks

**NOTICE**

The control characteristic name is based on the hooked variable name prefixed with `EH_ctrl_`.

Figure 3.10 illustrates a multiply hook; in this configuration the EHOOKS-DEV inserted hook is used as a multiplier of the original ECU value.

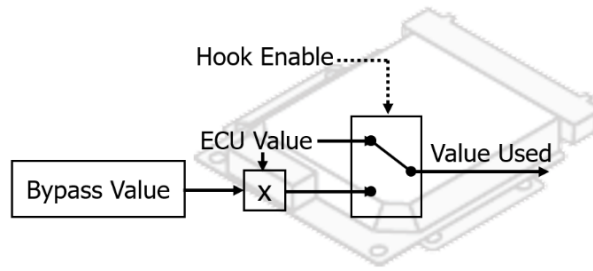


Figure 3.10: Multiply Hooks



**WARNING**

Offset and multiply hooks should not be used with hooks that are configured to use a forced-write. This is because when the hook is triggered at the forced-write location, there is no currently calculated ECU value to be used in the offset calculation and therefore the bypass value will simply be added or subtracted from 0



**WARNING**

Offset and multiply hooks should not be used with ECU variables which have non-linear `COMPU_METHODS` or linear `COMPU_METHODS` which do not intersect the origin (i.e. a `COMPU_METHOD` of  $y = mx + c$  where  $c$  is not 0). Currently EHOOKS-DEV does not check the `COMPU_METHODS` of variables selected for hooking.

3.2.2.5 Backup Measurement Copies

For each hook configured, EHOOKS-DEV allows the creation of a new measurement variable to store the original ECU-calculated variable value. Figure 3.11 illustrates a backup measurement copy. This allows measurement and easy comparison of the bypass value and the original ECU value via a measurement tool (e.g. INCA).

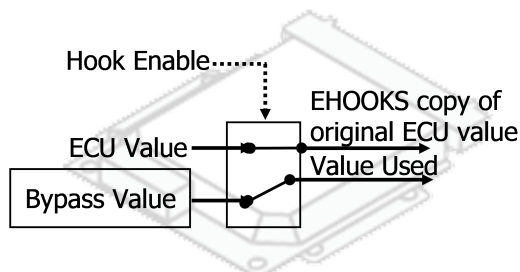


Figure 3.11: Backup Measurement Copy

Additionally, backup measurement copies can be used as input arguments to on-target bypass functions. This allows simple implementation of new algorithms that refine the existing ECU calculation of a variable, e.g. calculate an offset, add noise, implement filtering, etc.



## 3.2.2.6 Forced-Write Mechanisms

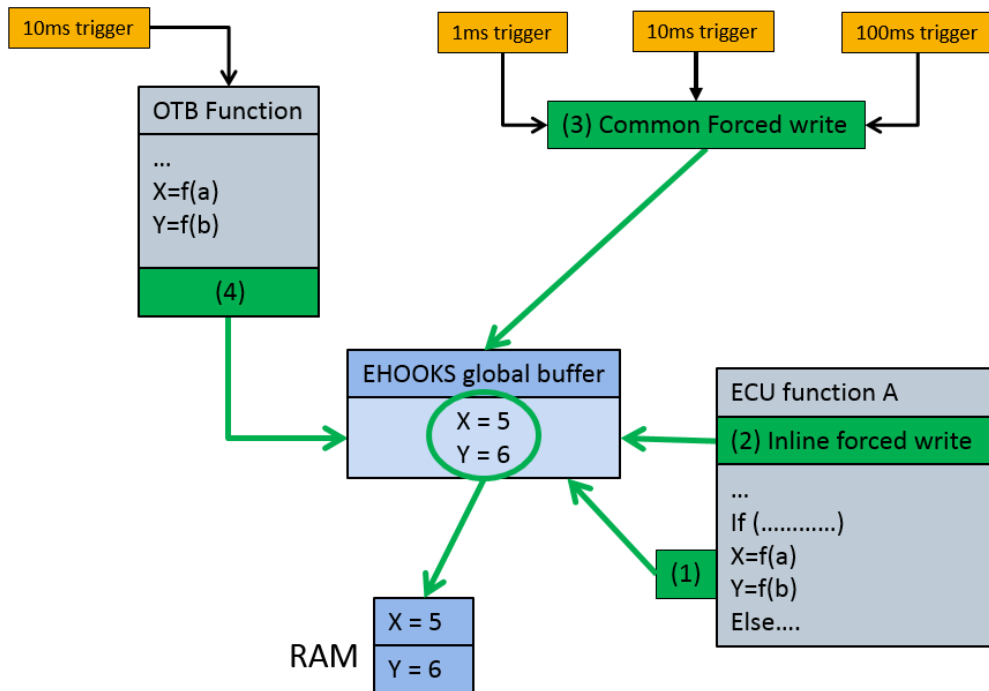


Figure 3.12: EHOOKS Forced writes mechanisms

Figure 3.12 shows the EHOOKS forced-write mechanisms. In this example, the default EHOOKS behaviour (Mechanism 1 in figure 3.12) is for the hooked variables X and Y to be updated with the current bypass value at the point where the ECU variable would have originally been written by the delivered ECU software (in 'ECU Function A'). The bypass values of X and Y are taken from the EHOOKS global buffer and written to the addresses for X and Y in RAM.

In addition to this default mechanism, EHOOKS-DEV allows the configuration of Forced-write mechanisms whereby the hooked ECU variable is forcibly updated with the bypass value. Forced-writes can be configured in 3 ways:

1. *Inline Forced-Writes (see 5.2.2 Configuring Properties of a Variable Hook)*

An inline forced-write means that the bypass value for a hooked variable will be updated at the beginning of each ECU process which contains a write to the original ECU variable (Mechanism 2 in figure 3.12).

2. *Common Forced-Writes (see 5.2.2 Configuring Properties of a Variable Hook)*

Common forced-writes allow the selection of any bypass containers to be used to update the hooked ECU variable with the current bypass value (Mechanism 3 in figure 3.12)

3. *Forced-writes at the output of On-Target bypass function dispatch point (see 10.2.2 EHOOKS ECU Trigger Source Block)*

Forced-writes can also be applied to on-target bypass functions. In the context of an on-target bypass function a forced-write means that, directly after the on-target bypass function has executed, the calculated values will be immediately and forcibly written into the ECU measurement variables configured as outputs to the on-target bypass

function (Mechanism 4 in figure 3.12).

These forced-write mechanisms can be very helpful to enable a successful bypass experiment, and can help in the following circumstances:

- Hook code not being executed

In some situations the EHOOKS-DEV place hooks aren't executed on the ECU. There are two common situations where this can be the case:

1. Certain processes/rasters are not being executed within the ECU, perhaps due to the ECU being used in a reduced hardware environment (i.e. not being used within the vehicle) or due to certain processes/rasters only executing during certain ECU modes of operation (e.g. during initialization, shutdown, limp-home-mode, etc)
2. Conditional code means that the original ECU variable write instruction is not being executed. As it is this instruction that EHOOKS uses to place the hook code, it will mean that the hook code also won't be executed.

In such situations, if an indicator measurement variable is configured for the hook (see section 3.2.2 EHOOKS-DEV Hook Configuration Properties – Indicators) then it becomes easy to observe that the hook code is not being executed. If the indicator variable is not updated to mirror the value of the hook enabler, this is a clear sign that the hook code is not being executed and that a forced-write configuration may be necessary.

- Need to change resolution/update frequency

For some experiments it may be necessary to increase the frequency at which a specific ECU variable is updated with the bypass value. This effectively allows a signal's resolution to be increased. By selecting a forced-write within a specific bypass container, the resolution of the hooked ECU variable can be controlled.

### 3.2.2.7 Safety Checks

For each hook configured, EHOOKS-DEV allows the creation of safety check code to monitor the hook for run-time errors. This safety check code detects some failures of the EHOOKS-DEV-inserted hooks (for example, if a hook placement has failed to patch all writes to an ECU variable). When triggered, this safety detection code will flag the error using a measurement and automatically disable the specific hook.

---

#### **NOTICE**

*The safety check failure measurement name is based on the hooked variable name prefixed with EH\_err\_xxx. An error is indicated when this measurement has the value 0x12. The safety check shutoff override calibration characteristic name is based on the hooked variable name prefixed with EH\_eovd. Setting this characteristic to true will allow the hook to continue to function even if the safety checks have failed.*

---



---

#### **NOTICE**

*The safety check feature requires additional ECU resources (code space and RAM) and therefore it is advisable to use the feature sparingly to avoid disrupting the original ECU behaviour.*

---

### 3.2.2.8 Default Project Configuration Settings

The **Tools** -> **Options** menu can be used to configure default settings for all EHOOKS projects. Using the **Default Settings** tab in the **Options** dialog, the EHOOKS-DEV user can configure default settings for the project and tab-specific settings shown in figure 3.13. When the user configuration is complete, clicking the **Save** button will save and apply this default configuration to the current project and all subsequently created EHOOKS projects. Clicking the **Restore Default** button will cancel these user default settings and restore the default configuration to it's previous state.

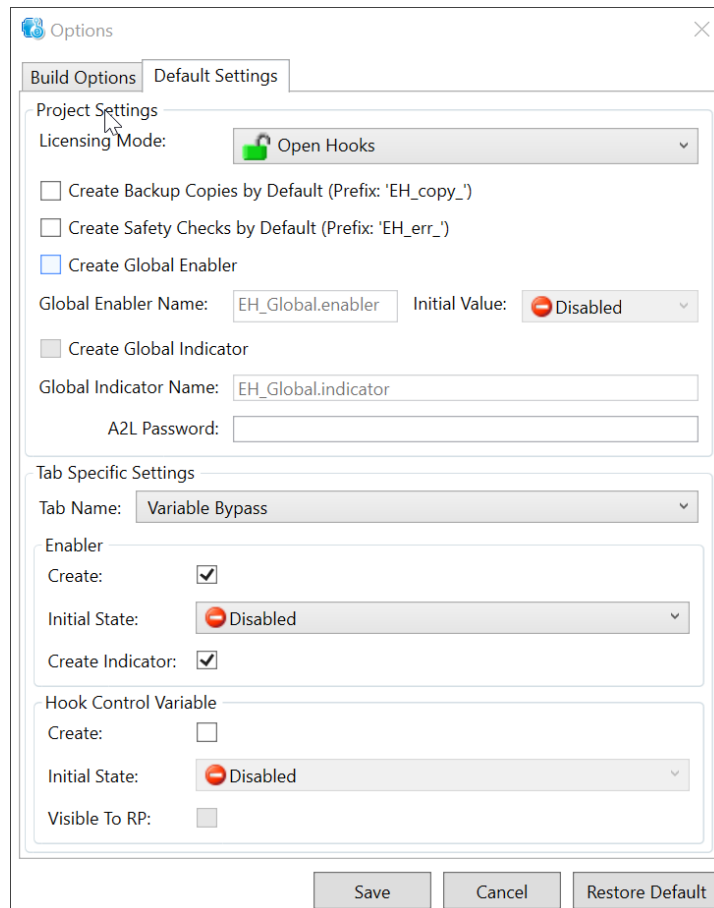


Figure 3.13: Setting default project configuration options

---

#### NOTICE

*The default settings are not saved with the EHOOKS project file, rather they are saved globally as part of the EHOOKS-DEV tool settings. They will therefore remain the same between different projects unless specifically changed within this dialog.*

---

### 3.2.3 Service Points

EHOOKS provides support for bypassing an original ECU process or function group with a Service Point to enable Service Based Bypass experiments to be performed. EHOOKS also allows run-time control of whether the original ECU process or the Service Point is executed. A schematic representation of how Service Points work is shown in figure 3.14.

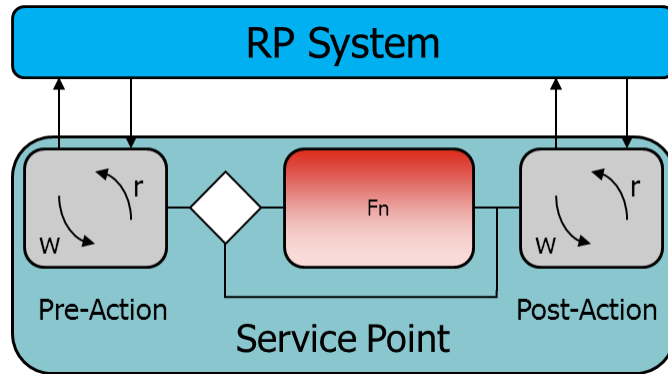


Figure 3.14: Service Based Bypass functionality

The Service Point can be used to send data to, and receive data from an external rapid prototyping system before and after the original ECU function. Wait-points can also be configured for the service point to allow for additional round-trip times to and from the RP system, and the original ECU function can be configured to be either executed or skipped.

Configuration of the Service Point properties must be performed from within INTECRIO or ASCET RP (service point configuration **cannot** be performed by EHOOKS). Please refer to the INTECRIO or ASCET RP user manual for details of how to do this.

---

**NOTICE**

*EHOOKS provides support for SBB V2.0, V2.1, V3.0 and V3.1*

---



---

**NOTICE**

*Some types of ECU do not provide support for SBB. You should check with your ECU supplier that SBB is supported before attempting to implement SBB with EHOOKS*

---

## 4 EHOOKS-PREP Dependencies

As seen in figure 3.1 (see section 3.1 [EHOOKS Workflow](#)), the first step in using EHOOKS is to receive an EHOOKS-prepared ECU software delivery. The functionality available with EHOOKS-DEV is somewhat determined by this preparation step. The following features of EHOOKS-DEV are controlled or influenced by the specifics of the EHOOKS preparation steps:

- **Variable Hooks**

The variables EHOOKS-DEV can hook within the ECU are determined as part of the ECU preparation. Each hook-type and hook-property can be used freely with all variables that have been allowed for EHOOKS hooking during the ECU preparation.

Access to additional variables for hooking can only be achieved by asking the ECU software provider to allow access and deliver a new EHOOKS-prepared ECU A2L file.

- **Function Bypass Hooks (*Process Hooks*)**

The ECU processes that EHOOKS can hook are determined as part of the ECU preparation. During this ECU preparation the ECU software provider can control whether an ECU process can be hooked and whether the ECU process should be visible within the EHOOKS-DEV GUI. If an ECU process can be hooked but is not made visible to the EHOOKS-DEV GUI, you must know the full name of the ECU process to be able to add a hook for it.

Access to additional ECU processes for hooking can only be achieved by asking the ECU software provider to allow access and deliver a new EHOOKS-prepared ECU A2L file.

- **External Bypass**

As part of the ECU preparation process, the ECU supplier can optionally specify whether it is possible for EHOOKS to create external bypass hooks.

Creation of external bypass hooks by EHOOKS is therefore only possible by asking your ECU supplier to support this.

- **Service Points**

The ECU processes and Function Groups that EHOOKS can bypass with a Service Point are determined as part of the ECU preparation. During this ECU preparation the ECU software provider can control whether an ECU process/Function Group can be hooked for this purpose and whether the ECU process/Function Group should be visible within the EHOOKS-DEV GUI. If an ECU process/Function Group can be hooked but is not made visible to the EHOOKS-DEV GUI, you must know its full name to be able to add a hook for it.

Access to additional ECU processes/Function Groups for hooking can only be achieved by asking the ECU software provider to allow access and deliver a new EHOOKS-prepared ECU A2L file.

- **Service Based Bypass over XCP**

As part of the ECU preparation process, the ECU supplier can optionally specify whether it is possible for EHOOKS to support external service based bypass over an XCP connection. If this option is not specified then external service based bypass is only possible over an ETK connection.

- **Forced Writes**

The ECU processes available within EHOOKS for forced writes are determined as part of the ECU preparation. During this ECU preparation, the ECU software provider can control whether an ECU process can be used and whether the ECU process should be visible within the EHOOKS-DEV GUI. If an ECU process can be used but is not made visible to the EHOOKS-DEV GUI, you must know the full name of the ECU process to be able to use it as a process within which to force a write.

Access to additional ECU processes for forced writes can only be achieved by asking the ECU software provider to allow access and deliver a new EHOOKS prepared ECU A2L file.

- **On-Target Bypass Code Bypass Containers**

The ECU processes available within EHOOKS for calling on-target bypass code are determined as part of the ECU preparation. During this ECU preparation, the ECU software provider can control whether an ECU process can be used and whether the ECU process should be visible within the EHOOKS-DEV GUI. If an ECU process can be used but is not made visible to the EHOOKS-DEV GUI, you must know the full name of the ECU process to be able to use it as a process within which to call on-target bypass code.

Access to additional ECU processes for calling on-target bypass code can only be achieved by asking the ECU software provider to allow access and deliver a new EHOOKS prepared ECU A2L file.

- **Post dispatch of On-Target Bypass Code using Bypass Containers**

During the ECU preparation process, the ECU supplier can specify bypass containers that can be used to dispatch on-target bypass code after the execution of the bypass container in addition to the default behavior which is to dispatch the on-target bypass code before the bypass container.

- **EHOOKS Memory Within ECU**

The available ECU memory which may be used by EHOOKS when creating hooked ECU software is determined as part of the ECU preparation. During each EHOOKS build, the available, used and remaining EHOOKS memory space within the ECU will be reported.

Access to additional ECU memory for EHOOKS can only be achieved by asking the ECU software provider to allocate more memory and deliver a new EHOOKS prepared ECU A2L file (and, optionally, a new ECU HEX file).

Resource usage of EHOOKS can be reduced by reducing the configuration complexity – reducing the number of hooked ECU variables and eliminating unnecessary indicator variables, enablers, etc.

- **Array Variables**

Sometimes, variables implemented as arrays in the ECU software are described in the A2L file as a number of scalar variables. In this case EHOOKS may not be able to successfully hook the variables unless the ECU software provider provides additional information in the EHOOKS preparation to indicate that these variables are actually part of an array.

- **On-Target Access to Scalar Characteristics**

The ECU software provider must provide specific support to enable EHOOKS to make

existing scalar calibration characteristics available for use within on-target bypass code/models. If such support is not included, EHOOKS-DEV will attempt to provide a default implementation, but this may not work in the context of a specific ECU.

Change to this support can only be achieved by asking the ECU software provider to provide such support and deliver a new EHOOKS-prepared ECU A2L file.

- **Special Purpose RAM**

Any ECU RAM areas that EHOOKS can use for special purposes (such as NVRAM) are determined as part of the ECU preparation. During this ECU preparation the ECU software provider can define which areas of RAM can be used for special purposes along with associated regular expressions that can be used in EHOOKS-DEV to define which variables are placed into the special purpose RAM areas.

*Note:* To allow the ASCET non-volatile flag feature to be used by EHOOKS, a special purpose RAM section must be defined by the ECU supplier explicitly as an NVRAM section as part of the ECU preparation.

Changes to the amount and type of special purpose RAM available for use by EHOOKS can only be achieved by asking the ECU software provider to provide this and deliver a new EHOOKS-prepared ECU A2L file.

- **Binary hex/s19 file patching**

As part of the ECU preparation process, the ECU software supplier can optionally define additional data to be patched at a specified address into the hooked hex/s19 file when it is built by EHOOKS-DEV. This information can be useful to help the EHOOKS-DEV user in identification and management of hooked hex/s19 files.

The types of data that can be inserted into the hooked hex/s19 file in this way are:

- A byte with a specified hex value
- The date in YYYY-MM-DD format
- The day
- The month
- The year
- The time in HH:MM:SS format
- The hour
- The minute
- The second
- The EHOOKS-DEV user's name
- The EHOOKS-DEV version
- A description field provided by the ECU software supplier
- The name of the EHOOKS-DEV configuration XML file
- The name of the input A2L file
- The name of the input binary hex/s19 file
- The name of the output A2L file
- The name of the output binary hex/s19 file
- The project description from the EHOOKS-DEV project

Patching any of the above data into the hooked hex/s19 file can only be achieved by asking the ECU software provider to enable this and deliver a new EHOOKS-prepared ECU A2L file

- **Save Password option**

During the ECU preparation process, the ECU supplier can permanently disable the 'Save Password' option in the EHOOKS-DEV user interface. If this option is disabled, it is not possible for an EHOOKS-DEV user to save the a2l file's password in the EHOOKS project configuration file, and they must therefore manually enter the password every time a project using the a2l file is loaded.

This feature can only be achieved by asking the ECU software provider to provide such support and deliver a new EHOOKS-prepared ECU A2L file.

- **Additional A2L information**

During the ECU SW preparation process, the ECU supplier has the option to insert a comment in the A2L file produced by the EHOOKS-PREP tool. When this option is set, it will also cause a comment to be inserted into the A2L file created by EHOOKS-DEV. EHOOKS-DEV will then insert information into the A2L file about the time and date of the EHOOKS-DEV run, any variables which have been hooked, any On-Target Bypass functions and any service points along with the description entered into the EHOOKS-DEV front-end tool. This functionality is provided such that A2L files are easily identifiable by having the EHOOKS information near the top of the file.

This feature can only be achieved by asking the ECU software provider to provide such support and deliver a new EHOOKS-prepared ECU A2L file.

- **ECU Function Calling**

During the ECU SW preparation process, a list of available ECU functions that can be called from a Simulink or ASCET model is defined by the ECU software supplier as part of the EHOOKS preparation process. Calling ECU functions from a Simulink or ASCET model can only be achieved by asking the ECU software provider to enable this functionality during the preparation of the ECU SW for use with EHOOKS.

This feature can only be achieved by asking the ECU software provider to provide such support and deliver a new EHOOKS-prepared ECU A2L file.

- **EPK Patching**

During the ECU SW preparation process, the ECU supplier can specify if, and how, the EPK information should be patched when an EHOOKS-DEV build is performed. If specified, matching information will be patched into both the hooked a2l and hex files. This patched information can then be used to easily identify matching a2l and hex files that are loaded into INCA.

This feature can only be achieved by asking the ECU software provider to provide such support and deliver a new EHOOKS-prepared ECU A2L file.

- **Embedded Build Scripts**

During the ECU SW preparation process, the ECU software supplier can optionally embed build scripts into the A2L file. These scripts can later be run by the EHOOKS-DEV user to perform pre-build and/or post-build actions. This feature can only be achieved by asking the ECU software provider to provide such support and deliver a new EHOOKS-prepared ECU A2L file.



## 5 Configuring EHOOKS-DEV

The EHOOKS-DEV Front-End is a graphical configuration tool that makes adding hooks into the ECU software as simple as selecting some variables from a list and pressing the build button.

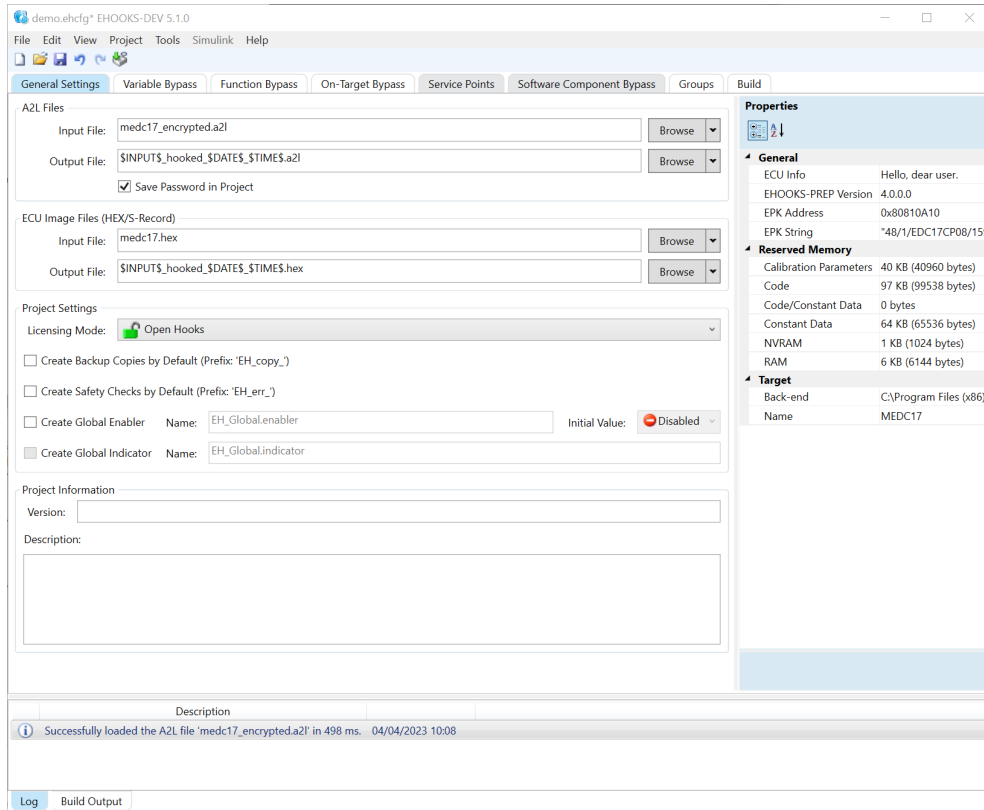


Figure 5.1: EHOOKS-DEV Front-End

When the EHOOKS-DEV Front-End is initially launched it will be started with a new empty EHOOKS-DEV project configuration. The project configuration can be saved by selecting **File** -> **Save** and selecting a location and file name for the project configuration. A new project can be created at any time by selecting **File** -> **New**. An existing project can be reloaded by selecting **File** -> **Open**.

The EHOOKS-DEV Front-End has a tabbed user interface. The tabs give a typical workflow through the EHOOKS-DEV configuration process:

- **General Settings**

Allows the basic EHOOKS-DEV project settings to be configured, such as the input ECU HEX and A2L files, project-wide default settings and general project information.

- **Variable Bypass**

Allows the selection of ECU variables that are to be hooked by EHOOKS-DEV and allows the setting of the hook properties.

- **Function Bypass**

Allows the selection of ECU functions (or processes) that will be hooked by EHOOKS-DEV.

- **On-Target Bypass**

Allows the creation and configuration of on-target bypass functions.

---

**NOTICE**

*If ASCET or Simulink are used to create the on-target bypass function, then the ASCET-SE EHOOKS Target or the EHOOKS-DEV Simulink Integration package are used to create the on-target bypass function. In these cases the on-target bypass function should not be configured directly within the EHOOKS-DEV Front-End, but rather within the ASCET-SE EHOOKS Target or the EHOOKS-DEV Simulink Integration package.*

*For details of using ASCET for on-target bypass with EHOOKS, please refer to the ASCET-SE EHOOKS Add-on User Guide for full details (this can be found in the ASCET installation directory within the folder <ASCET\_INSTALL\_DIRECTORY>\target\trg\_ehooks\documents).*

*For details of using Simulink for on-target bypass with EHOOKS, please see section [10 Creating and Working with On-Target Bypass from Simulink](#).*

---

- **Service Points**

Allows the creation of service points in the ECU software for external bypass experiments

- **Groups**

Allows the creation of groups of hooks which can then be enabled and disabled together at run-time.

- **Build**

Allows the build options to be configured.

The EHOOKS-DEV Front-End creates an EHOOKS configuration file (with the extension .ehcfg) which is a standardized XML file. For details of the XML configuration file format for EHOOKS-DEV please see section [11.7.1 EHOOKS-DEV Project Configuration File](#).

## 5.1 General Settings Tab

Various general project-wide settings can be configured from within the General Settings tab of the EHOOKS-DEV Front-End. This tab is divided into four sections – A2L, ECU Image, Project Settings and Project Information.

### 5.1.1 A2L Section

The A2L section allows the input and output A2L files to be used by EHOOKS-DEV to be configured. Clicking on the **Browse** button brings up a standard file-selection dialog allowing selection of A2L files.

Figure 5.2: Input and Output A2L Files

The input (and output) A2L files can also be quickly configured by dragging and dropping from Windows Explorer. The input A2L file must include the additional EHOOKS information

included by the ECU software provider when they prepared the ECU software for use with EHOOKS (see section 3.1.1 Step 1: EHOOKS Prepared ECU Software). This additional EHOOKS information will typically be embedded inside the A2L file. However, the ECU software provider may optionally provide this information in a separate EHOOKS ECU internals file. EHOOKS-DEV will prompt for a password when importing the EHOOKS information – this password should be obtained from the ECU software provider.

---

**NOTICE**

Where the EHOOKS information is provided in a separate EHOOKS ECU internals file this will have an extension `.ECUINTERNALS`. To allow EHOOKS-DEV to load the A2L file and its associated ECU internals file it is necessary that both the A2L file and the ECU internals file have identical file names and that only the file extensions differ.

---

If the small drop-arrow on the **browse** button is selected, a pop-up menu allows the selection of whether the A2L files are specified via absolute path (which is the default behaviour of the **browse** button) or a relative path (relative to the saved EHOOKS project configuration file or, if specified, the `--base-dir` command-line option to EHOOKS-DEV command-line tools). See figure 5.3.

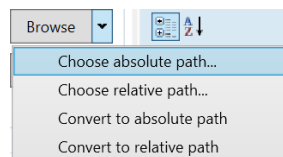


Figure 5.3: Absolute and Relative Path Specification

Selecting to configure the A2L files via a relative path brings up the dialog shown in figure 5.4. The relative path can be configured using `'.'` to refer to the current directory and `'..'` to refer to the parent directory. As the relative path is entered, EHOOKS will automatically complete it based on the file names available within the file system.

---

**NOTICE**

Before the **Choose relative path...** command can be selected, the project configuration must first be saved. If the option is disabled, check that the project has been saved.

---

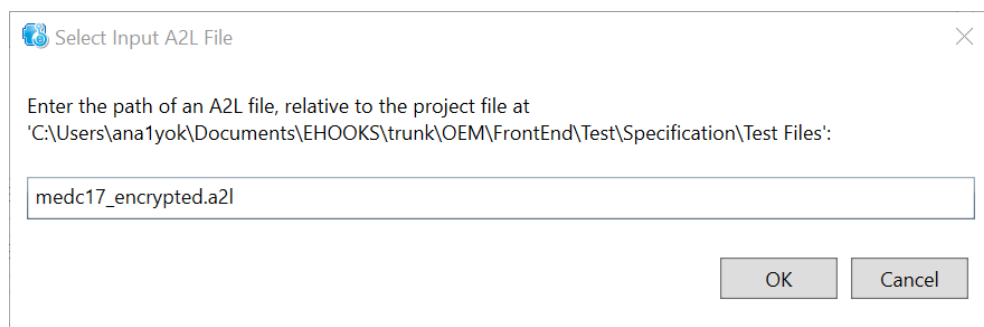


Figure 5.4: Adding an A2L file via a Relative Path

To provide help with the management of multiple generated files, EHOOKS provides three

macros, `$INPUT$`, `$DATE$` and `$TIME$`, that can be inserted into the output filenames using both the relative or absolute path dialogs. When these macros are used, `$INPUT$` is replaced by the input filename, `$DATE$` is replaced by the current date in YYYYMMDD format, and similarly `$TIME$` is replaced by the current time in HHMMSS format. The example shown in figure 5.4 would therefore result in an output filename that looks like `inputfilename_hooked_YYYYMMDD_HHMMSS.a2l`. The three macros can be inserted at any position in the filename to provide maximum flexibility with file naming conventions.

When the check-box **Save Password in Project** is checked, the EHOOKS A2L password is stored in the EHOOKS-DEV project configuration file and will not need to be re-entered when the EHOOKS-DEV project configuration file is re-loaded. If the password is not saved in the EHOOKS-DEV project configuration file, then it is necessary to provide the EHOOKS A2L file password as a command-line option when using EHOOKS-DEV from the command line. See section 11.1 [EHOOKS-DEV Command Line Usage](#), for details. When **Save Password in Project** is unchecked, the password is encrypted and stored locally on the PC, and the same user will not need to re-enter the password for the same project on the same PC.

### 5.1.2 ECU Image

The ECU Image section allows the input and output ECU HEX files to be used by EHOOKS-DEV to be configured. Clicking on the “Browse” button brings up a standard file-selection dialog allowing selection of ECU Hex files.



Figure 5.5: Input and Output HEX files

The input (and output) ECU hex files can also be quickly configured by dragging and dropping from Windows explorer. Relative paths can be configured for the ECU image in exactly the same way as described in the previous section for A2L files, and the `$INPUT$`, `$DATE$` and `$TIME$` macros can also be used to customize the output .hex filenames.

### 5.1.3 Project Settings

The Project Settings section allows a number of project-wide default options to be configured along with the associated names/naming prefixes:

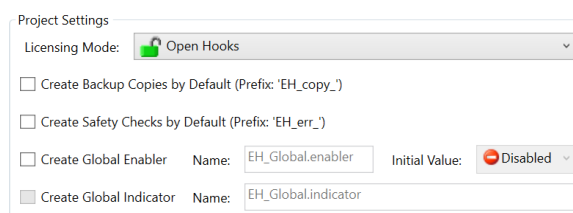


Figure 5.6: Default Project Settings

- Licensing Mode

This determines whether EHOOKS will generate ECU software that requires the usage of EHOOKS-CAL or EHOOKS-BYP to work with the EHOOKS-created hooks. If **Locked Hooks** mode is selected then the use of EHOOKS-CAL or EHOOKS-BYP is necessary in order to be able to use the hooks placed into the ECU software. If **Open Hooks** mode is selected then the hooked ECU software can be used independently of EHOOKS-CAL and EHOOKS-BYP. For more details see section 6 [Working with Hooked ECU Software in INCA](#).

**NOTICE**

*A special licence is required to be able to work with Open Hooks. Please contact your local ETAS sales region (see section [ETAS Contact Addresses]) for details.*

- Create Backup Copies by Default

If selected, EHOOKS-DEV will by default create backup measurement copies (see section [6.3 Backup Measurement Copies](#)) of all hooked ECU variables, unless specifically overridden in the hook configuration. The value of the project default for Create Backup Copies will be displayed in the Variable Bypass Tab

- Create Safety Checks by Default

If selected, EHOOKS-DEV will by default insert additional safety check code (see section [6.4 Safety Checks](#)) for all ECU variable hooks, unless specifically overridden in the hook configuration.

- Create Global Enabler

If selected, EHOOKS-DEV will create a global hook enabler (see section [3.2.2.1 Hook Enablers](#)). The initial value of the global hook enabler is set to disabled by default, which will cause all hooks to be turned off at ECU startup. The initial value of the global hook enabler can be changed to enabled from the Project Settings section, which will allow hooks to be turned on at ECU startup.

- Create Global Indicator

If selected, EHOOKS-DEV will create a global indicator measurement (see section [3.2.2.2 Indicators](#)).

#### 5.1.4 Project Information

The Project Information section allows a project version and description to be recorded in the EHOOKS-DEV configuration file. This information is not processed by EHOOKS-DEV in any way; it is simply stored in the EHOOKS-DEV configuration file. This may be useful for configuration management of projects or some other purpose.

The screenshot shows a form titled "Project Information". It contains two input fields: "Version:" followed by a text input box, and "Description:" followed by a larger text area for entering text.

Figure 5.7: Project Information

When the General Settings tab is selected and an input A2L file containing EHOOKS information is loaded, the EHOOKS-DEV property browser will show some basic information about the EHOOKS preparation performed by the ECU software provider. This includes:

- **ECU Info** Contains any message/documentation provided by the ECU software

developer about the EHOOKS preparation as a string. If the string is long then it is possible to cut-and-paste the text into another application to make it easier to read.

- **EHOOKS-PREP Version** The version number of EHOOKS-PREP used to prepare the ECU software for use by EHOOKS-DEV.

### EPK address and string

If the A2L file contains an EPK string, the address and value of the string are displayed. EHOOKS uses the EPK string information in 2 ways. Firstly, it will raise a warning message if a2l and hex files are loaded into EHOOKS for which the EPK strings do not match. Secondly, when merging DCM/hex/s-record files, EHOOKS uses the EPK string for consistency checking as described in section [5.8.1 Configuring Build Source Files](#).

- **Reserved Memory:**

- **Calibration parameters:** The amount of free memory reserved in the prepared ECU software for calibration parameters created by EHOOKS.
- **Code:** The amount of free memory reserved in the prepared ECU software for code created by EHOOKS and used for on-target bypass.
- **Constant Data:** The amount of free memory reserved in the prepared ECU software for constant data created by EHOOKS and used for on-target bypass.
- **Code/Constant Data:** The amount of free memory reserved in the prepared ECU software for code and/or constant data created by EHOOKS and used for on-target bypass.
- **RAM:** The amount of free memory reserved in the prepared ECU software for (RAM) variables created by EHOOKS and used for on-target bypass.

- **Target:**

- Back-end

Gives the path to the EHOOKS-DEV Back-End installation that will be used to process the configuration.

- Name

Gives the name of the EHOOKS-DEV Back-End ECU Port being used with the configuration.

---

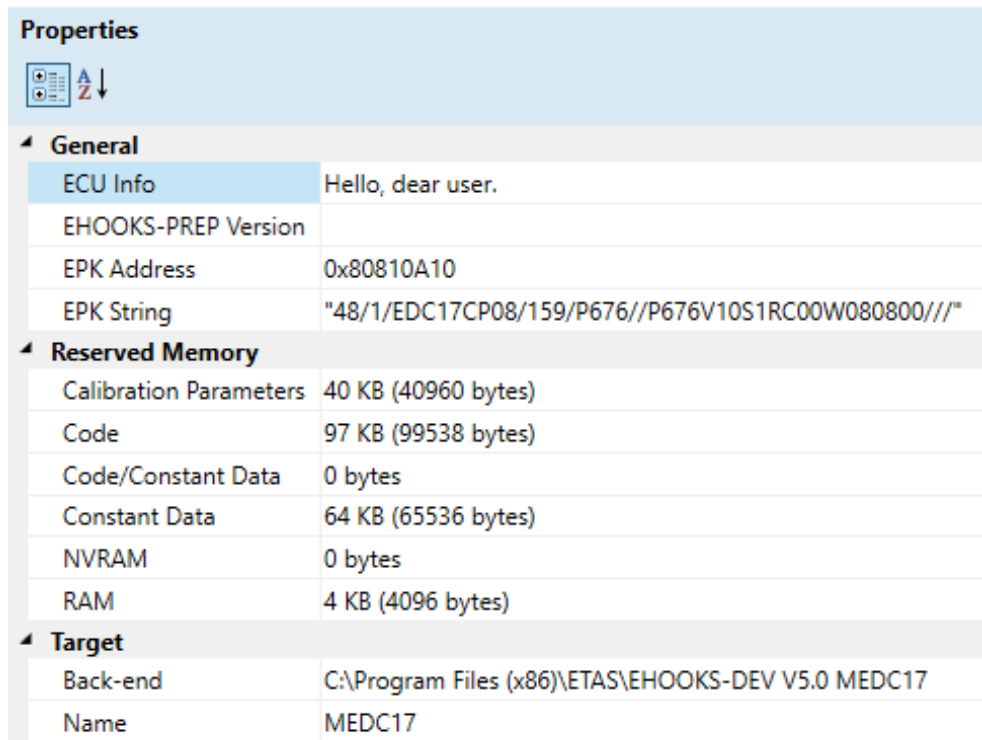
*NOTICE*

*The target information is determined automatically from the loaded input A2L file and cannot be changed.*

---

## 5.2 Variable Bypass Tab

Using the Variable Bypass Tab, ECU variables can be selected to be hooked and the hook configuration properties can be set. This section details how to set the configuration options. For details on the functionality of each of the configuration items please see section [EHOOKS-DEV Features](#), and for details on how to work with hooked ECU software created with the various configuration options please see section [6 Working with Hooked ECU Software in INCA](#).




Properties	
	
<b>General</b>	
ECU Info	Hello, dear user.
EHOOKS-PREP Version	
EPK Address	0x80810A10
EPK String	"48/1/EDC17CP08/159/P676//P676V10S1RC00W080800//"
<b>Reserved Memory</b>	
Calibration Parameters	40 KB (40960 bytes)
Code	97 KB (99538 bytes)
Code/Constant Data	0 bytes
Constant Data	64 KB (65536 bytes)
NVRAM	0 bytes
RAM	4 KB (4096 bytes)
<b>Target</b>	
Back-end	C:\Program Files (x86)\ETAS\EHOOKS-DEV V5.0 MEDC17
Name	MEDC17

Figure 5.8: EHOOKS-Prepared ECU Properties shown in the General Settings Tab

The interface for adding and configuring an ECU variable hook follows a simple three step workflow.

#### How to add and configure an ECU variable hook

- Step 1: Select the ECU variable to be hooked (see red highlight in figure 5.9). Selecting an ECU variable to hook will cause EHOOKS-DEV to add it to the list in step 2.
- Step 2: Provides a list of ECU variables selected to be hooked within the EHOOKS-DEV project configuration (see blue highlight in figure 5.9). Selecting a hooked variable (or several hooked variables) from this list allows the associated hook properties to be configured in step 3. To remove a variable hook from the configuration simply select it and press the Delete key or **right-click -> Delete**.
- Step 3: Configure the properties to provide the desired ECU variable hook information for the selected ECU variable hooks (see green highlight in figure 5.9)

##### 5.2.1 Selecting Variables to Be Hooked

The Variable Bypass Tab provides two ways to add an ECU variable to the list of hooked variables: a variable section dialog and a quick-add field.

Selecting the **Variable Selection...** button will display a new variable selection dialog (figure 5.10). The left-hand column allows the displayed variable list to be filtered by the A2L file function groups (including filtering by the input, output and local measurements to a function). To quickly move to the desired function group, click on an entry in the function group list to move focus and then begin typing.

The middle-column displays a variable list which can be filtered by typing a filter string into

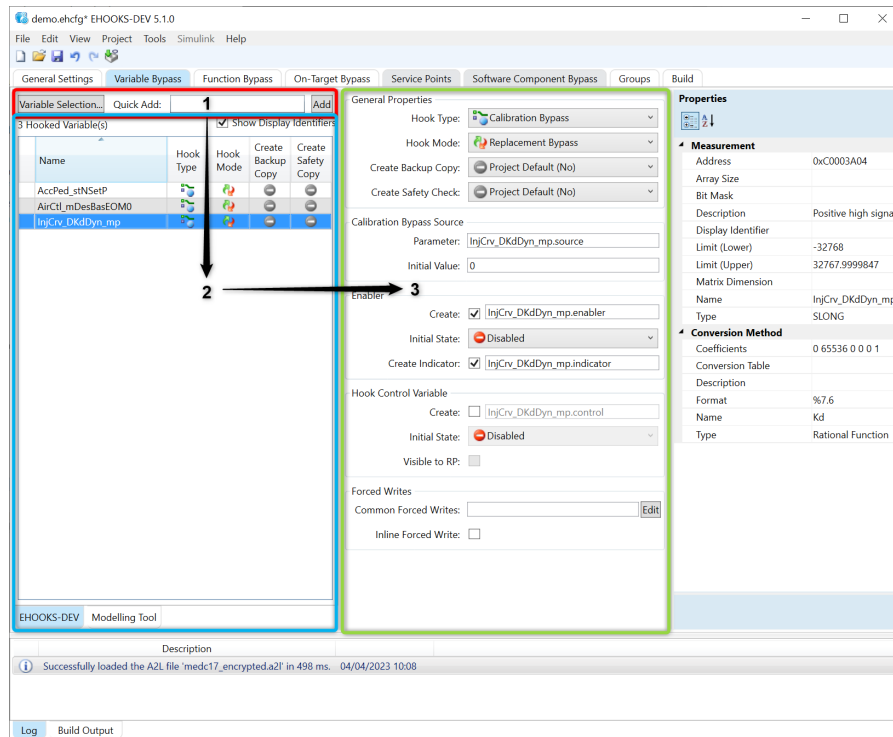


Figure 5.9: Variable Bypass Configuration

the text box. The filter string can include the wildcard characters `?` and `*`. `?` will match any single character and `*` will match any number of characters. Note that the filter string contains an implicit `*` wildcard at the end of the string. The character `$` can be used to match the end of a variable name. As the filter string is updated, the variable list will dynamically update to show any matching ECU variables that can be hooked.

The right-hand column shows the ECU variables that have been selected for hooking. ECU variables can be moved in and out of this list using the `>>` and `<<` buttons, or by pressing `Ctrl+right-arrow` and `Ctrl+left-arrow`, respectively. Additionally, double clicking on an entry in either list will move it to the other list. If an array variable has been selected for hooking then the elements of the array to be hooked can be configured by clicking on the `...` button, the number of hooked elements is then shown in the right-hand column. Multiple elements can also be quickly selected or deselected by using either `Shift+click` to select a block of variables or array elements or `Ctrl+click` to select multiple individual variables.

The **Show Display Identifiers** check box allows the list of variable names to be changed to show the `DISPLAY_IDENTIFIER` fields from the A2L file.

The variable selection dialog (figure 5.10) also supports the **Export** and **Import** of lists of variables to be hooked. This feature is useful for either reusing a list of hooked variables in more than one EHOOKS project, or for exporting lists of hooked variables to an INCA experiment. Clicking on the **Export** button brings up a new Windows Explorer dialog that allows the currently selected list of variables to be exported to an EHOOKS-DEV Measurement (\*.mst) file or to a \*.lab file as used by INCA. Clicking the **Import** allows an existing \*.mst or \*.lab file to be imported. The variables listed in the imported file will then be added to the selected variables list in the variable selection dialog. A warning message will be displayed in the EHOOKS-DEV Log if any of the variables imported from the file are not available for hooking in loaded A2L file.



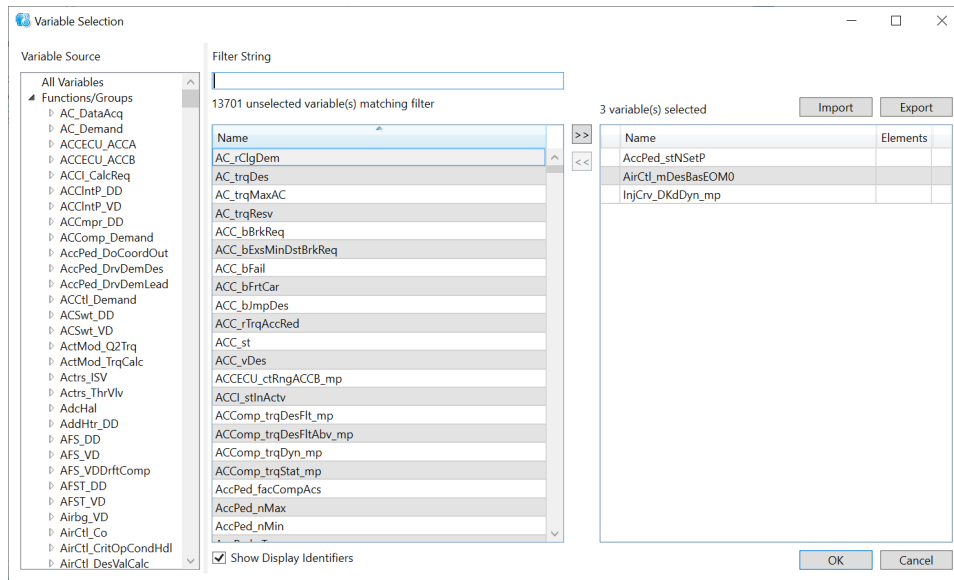


Figure 5.10: Variable Selection Dialog

The quick-add field allows you to directly enter the name of an ECU variable. EHOOKS-DEV will immediately offer auto-completion suggestions from the list of available ECU variables. Once the desired ECU variable has been selected, clicking the **Add** button (or pressing Enter) will move the variable into the list of ECU variables selected for hooking.

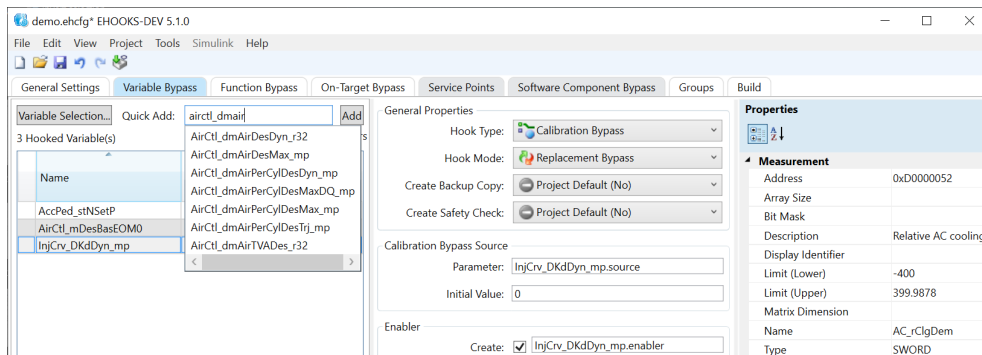


Figure 5.11: "Quick Add" A Variable Hook By Name

### 5.2.1.1 Viewing Write-Points for Variables

If your ECU software supplier has configured the A2L file to allow EHOOKS-DEV to show the write-points in the software for each variable, an additional view will be present in the variable bypass tab (figure 5.12).

This view will show, for each variable highlighted in the variable list, the points in the ECU software where that variable is written. Each write-point will show the name of the OS process writing to the variable (even if the write is actually performed by a sub-function).

Variable Write Points		<input type="checkbox"/> Display Copies	<input type="checkbox"/> Display Service Points
	Process		
▲	<b>VariableA</b>		
	Process_10ms		
	Process_20ms		
▲	<b>VariableB</b>		
	Process_10ms		
▲	<b>VariableC</b>		
	Process_100ms		
	Process_20ms		
▲	<b>VariableD</b>		
	Process_100ms		

Figure 5.12: Table showing variable write-points

You can view extra information about each write-point via the two checkboxes:

- "Display Service Points" will show the name of the service-point (if present) corresponding to each ECU process. This allows you to select that service-point in other tools such as INTECRIO.
- "Display Copies" will show the name of the message-copy variable written by the ECU process (if the write is not directly to the A2L measurement)

Text in each field can be copied to the clipboard for use elsewhere.

## 5.2.2 Configuring Properties of a Variable Hook

Once some ECU variables have been selected for hooking, they will appear in the hooked variable list within the Variable Bypass tab. Selecting a variable (or many variables) from within the **hooked variable** list allows its (their) properties to be configured on the right-hand side. For full details on the hook properties see section [3.2.1 EHOOKS-DEV Hook Types](#) and section [3.2.2 EHOOKS-DEV Hook Configuration Properties](#).

---

### NOTICE

The hooked variable list is actually two lists that can be viewed independently using the tab control at the bottom of the grid view control. The **EHOOKS-DEV** list contains a list of the ECU variables that have been selected for hooking and configured within the EHOOKS-DEV tool itself. The **Modeling Tool** list contains a list of the ECU variables that have been selected for hooking by an external modeling tool (such as ASCET or Simulink). The list of variables in the **Modeling Tool** list cannot be changed directly within the EHOOKS-DEV user interface; the EHOOKS configuration environment of the modeling tool should be used to achieve this. However, the configuration properties for these variable hooks can and should be set within the EHOOKS-DEV user interface.

---

### NOTICE

When a variable is selected the property pane displays information about the variable from the loaded A2L file.

---

The **Hook Type** dropdown allows the type of hook to be configured. Once the hook type is

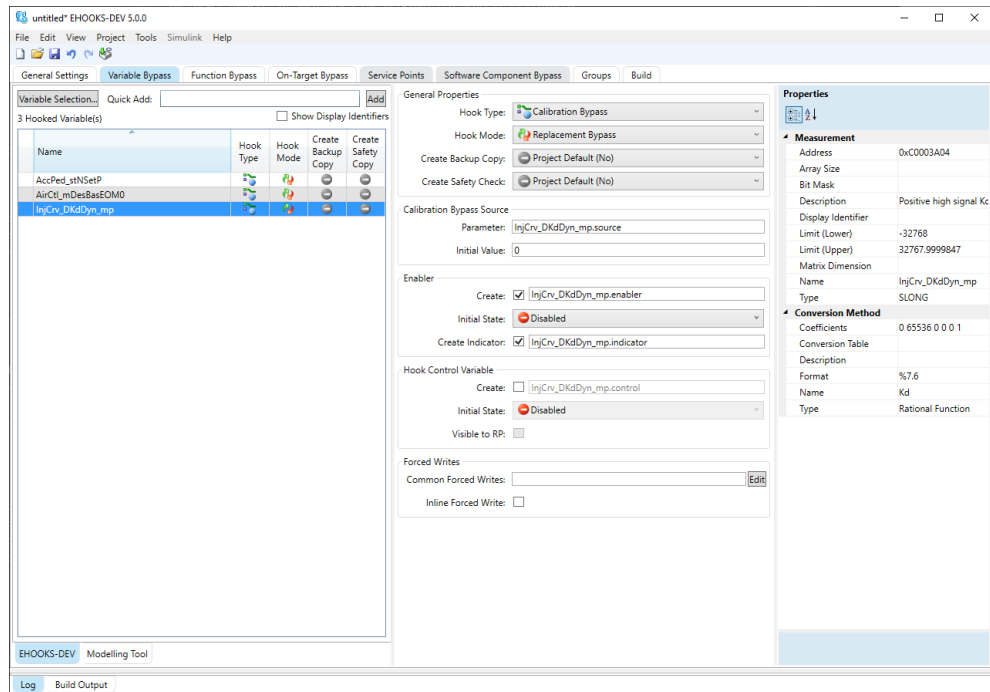
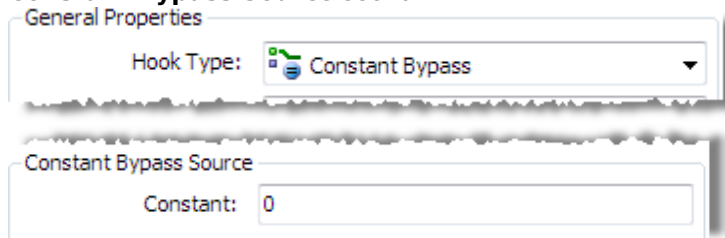


Figure 5.13: Setting Variable Hook Properties

configured the corresponding **Source** section should be defined. The hook type can be:

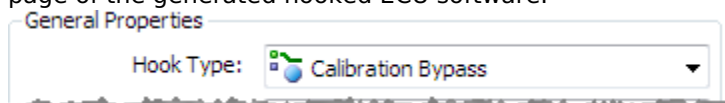
- **Constant bypass**

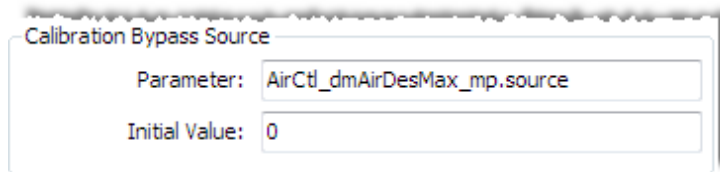
The hooked ECU variable will be bypassed by the fixed constant value configured in the **Constant Bypass Source** section.



- **Calibration bypass**

The hooked ECU variable will be bypassed by the current value of the newly created calibration characteristic defined in the **Calibration Bypass Source** section. This effectively allows calibration control over any ECU variables (measurements) that can be hooked with EHOOKS. In addition to defining the name of the newly created calibration characteristic to be used to bypass the hooked ECU variable, the initial value for this characteristic is configured. EHOOKS-DEV places this value into the calibration reference page of the generated hooked ECU software.





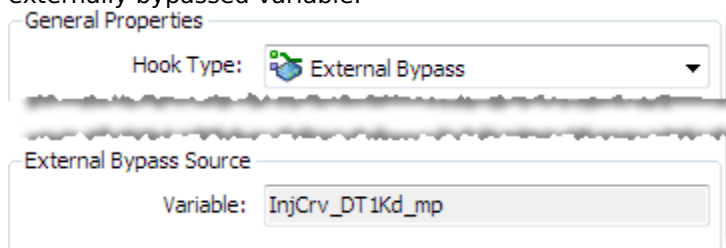
Calibration Bypass Source

Parameter: AirCtl\_dmAirDesMax\_mp.source

Initial Value: 0

- **External bypass**

The hooked ECU variable will be bypassed by a value calculated on external rapid-prototyping hardware. EHOOKS-DEV will add the necessary information to the hooked ECU A2L file so that when it is loaded into an external rapid prototyping tool (e.g. INTECRIO or ASCET-RP), the ECU variable can be set up for external bypass. For an external bypass hook, no information needs to be configured in the **External Bypass Source** section. This section will provide a read-only display of the A2L name of the externally bypassed variable.



General Properties

Hook Type: External Bypass

External Bypass Source

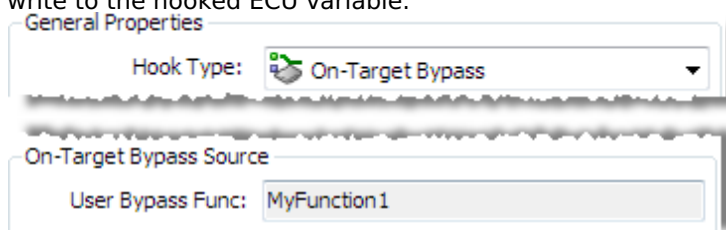
Variable: InjCrv\_DT1Kd\_mp

**NOTICE**

*If the ECU software prepared for EHOOKS by the ECU supplier does not support External Bypass then the External Bypass hook type will be greyed-out and cannot be selected.*

- **On-target bypass**

The hooked ECU variable will be replaced by a newly provided function as defined in the On-Target Bypass Tab. For an on-target bypass hook, no information needs to be configured in the **On-Target Bypass Source** section, as this information is configured within the On-Target Bypass Tab (see section [5.4 On-Target Bypass Tab](#)). This section will display a read-only display of the on-target bypass functions that are configured to write to the hooked ECU variable.



General Properties

Hook Type: On-Target Bypass

On-Target Bypass Source

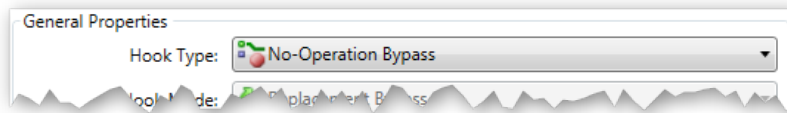
User Bypass Func: MyFunction1

**NOTICE**

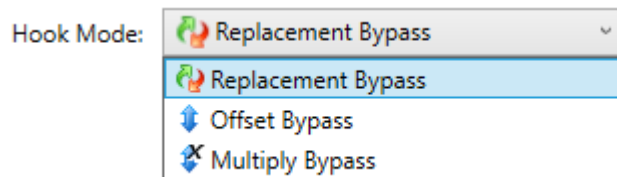
If the on-target bypass is being configured using ASCET or Simulink, the hook should not be added to the variable bypass tab manually. ASCET or Simulink should be used to add the corresponding hooks and configuration directly (see section [10 Creating and Working with On-Target Bypass from Simulink](#) or the ASCET-SE EHOOKS Add-on User Guide for full details – this can be found in the ASCET installation directory within the folder <ASCET\_INSTALL\_DIRECTORY>\target\trg\_ehooks\documents). These hooks will appear within the Variable Bypass Tab under the **Modeling Tool** list available at the bottom of the hooked variable list.

- **No-Operation bypass**

The hooked ECU variable will not be written to when No-operation bypass is configured. When the No-operation hook is enabled, all writes to the hooked variable from the ECU software are replaced with a No-operation instruction.



The **Hook Mode** drop down allows the way in which the bypass value is used by the hook to



be configured. The Hook Mode can be:

- **Replacement Bypass**

The configured bypass value will be used by the hook to completely replace (overwrite) the ECU calculation for the hooked ECU variable.

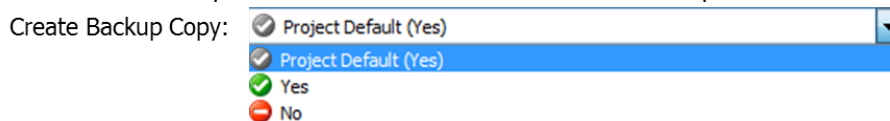
- **Offset Bypass**

The configured bypass value will be added or subtracted to the ECU calculation for the hooked ECU variable.

- **Multiply Bypass**

The configured bypass value will be multiplied with the ECU calculation for the hooked ECU variable.

The **Create Backup Copy** drop down allows the configuration of whether EHOOKS will create new ECU measurement variables to record a copy of the ECU calculation for the hooked ECU variables. The drop down allows the selection from three options:



- **Project Default**

The creation of the backup copy ECU measurements will be determined based on the

project-wide setting configured on the General Settings Tab (see section [5.1.3 Project Settings](#)). The Project Default value is displayed by the grey icon.

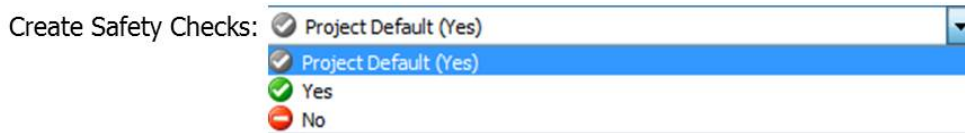
- **Yes**

This means EHOOKS will create a new measurement variable and keep it up to date with the most recent ECU calculation for the hooked ECU variable.

- **No**

This means EHOOKS will not create a backup copy measurement.

The **Create Safety Check** drop down allows the configuration of whether EHOOKS will add additional run-time code to monitor the run-time performance of the hook. This code will determine if the hook is running correctly and, if not, will disable the hook and indicate that an error has been detected. The drop down allows the section from three options:



- **Project Defaults**

The addition of safety-checking code will be determined based on the project wide setting configured on the General Settings Tab (see section [5.1.3 Project Settings](#)). This makes it quick and easy to change the setting for all the hooks together.

- **Yes**

This means EHOOKS will create safety checking code for the hooked ECU variable to monitor its performance at run-time.

- **No**

This means EHOOKS will not monitor the performance of the hooked ECU variable at run-time.

The **Enabler** settings allow the configuration of whether EHOOKS will add a calibration characteristic that can then be used at run-time to control whether the hook is activated or not. To configure an enabler, the checkbox must be ticked and then the name of the enabler calibration characteristic can be set in the text field. The initial value of the hook enabler is set to disabled by default, which causes the hook to be turned off at ECU startup. The initial value of the hook enabler can be changed to enabled, which will cause the hook to be turned on at ECU startup.

If an enabler is configured, it is also possible to configure an indicator. The **Indicator** setting allows the configuration of whether EHOOKS will add a measurement variable that will mirror the current value of the enabler characteristic at run-time. To configure an indicator, the checkbox must be ticked and then the name of the indicator measurement variable can be set in the text field.

A hook control variable can also be added to the configuration by ticking the **Create** checkbox in the relevant section and setting the name to be used for the control variable. The **Initial State** of the hook control variable is set to disabled by default, which causes the hook to be turned off at ECU startup. The **Initial State** of the hook control variable can be changed to enabled, which will cause the hook to be turned on at ECU startup. The hook control variable

The screenshot shows a configuration window with three main sections:

- Enabler:**
  - Create:  ACCECU\_ctRngACCA\_mp.enabler
  - Initial State:  Disabled
  - Create Indicator:  ACCECU\_ctRngACCA\_mp.indicator
- Hook Control Variable:**
  - Create:  ACCECU\_ctRngACCA\_mp.control
  - Initial State:  Disabled
  - Visible to RP:
- Forced Writes:**
  - Common Forced Writes: BypEHOOKs\_10ms\_Proc
  - Inline Forced Write:

Figure 5.14: Configuring Variable Bypass Enabler, Indicator, Hook Control Variables and Forced write

can also be configured to allow it to be updated from bypass software running on external rapid prototyping hardware by ticking the **Visible to RP** checkbox.

---

**NOTICE**

The configured control variable name will be created as a C variable that can then be managed at run-time by an on-target or external bypass function. Therefore the configured name should be a valid C identifier. However, EHOOKS will automatically convert the name into a valid C identifier if the name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (`__`). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.

The converted C name will need to be used in any on-target or external bypass function that manages a control variable at run-time.

---

The **Common Forced Write** box displays a read-only list of the configured ECU processes in which an update of the bypass value for the hooked ECU variable will be performed. The set of write locations can be updated by clicking the **Edit** button. This will display the **Select Forced Write Locations** dialog. In the left column a list of the available ECU processes is displayed. EHOOKS can be configured to force the update of the bypass value in one or more of these processes by moving them into the right column: select them and use the **>>** button. To remove a forced write location from the configuration, select it and click the **<<** button. The dialog offers a filter to help locate the desired ECU process; this behaves in the same way as variable selection (see section 5.2.1 [Selecting Variables to Be Hooked](#)).

The Elevated Permissions checkbox allows the hook to be written with elevated memory access permissions to bypass memory protection. The advanced project option for Elevated Permissions must be first enabled (see section 11.5.4 [Enable Elevated Permissions](#)).

In some situations the ECU software provider may have prepared some ECU processes to allow their use as forced write locations, but indicated that their names should not be

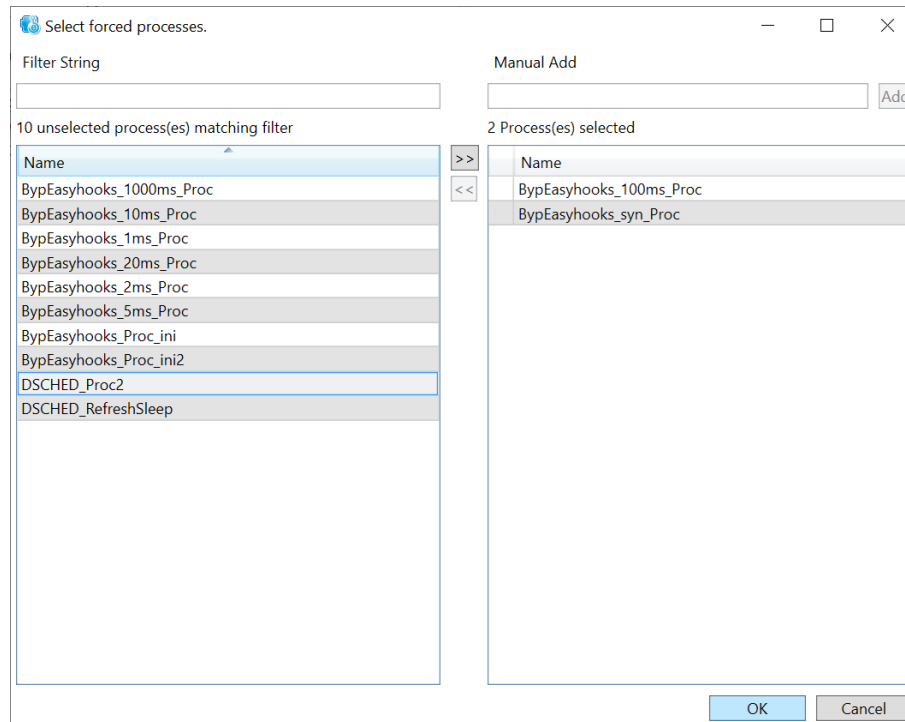


Figure 5.15: Selecting Forced Write Locations

displayed in the EHOOKS-DEV user interface. In this case, such forced write locations can be added to the configuration by typing their name into the **Manual Add** text field and clicking the **Add** button.

The **Inline Forced Write** check box allows the configuration of a forced write that EHOOKS will automatically insert at the start of every ECU process where the hooked ECU variable is updated within the original ECU software.

### 5.2.3 Multi-Select Operation

It is possible to select more than one hooked variable; this allows all the selected hooks' properties to be updated together. For properties that are the same for all selected hooks, the EHOOKS-DEV configuration tool will display the configured property. For properties that differ between the selected hooks, the EHOOKS-DEV configuration tool will display a blank selection. A special feature is available for the enabler, indicator and control variable names: if the naming convention is consistent for the selected hooks then the macro <name> is used to represent the name of the hooked A2L variable. It is possible to use the <name> macro when updating the desired enabler or indicator names and the EHOOKS-DEV configuration tool will automatically replace this with the A2L name of each hooked variable.

When multiple hooks are selected, changing a hook property will immediately apply the change to all selected hooks.

---

#### **NOTICE**

*EHOOKS-DEV provides full undo-redo support which can be very helpful if the use of the multi-select feature has accidentally caused an undesirable change of a property for a large number of hooks.*

---



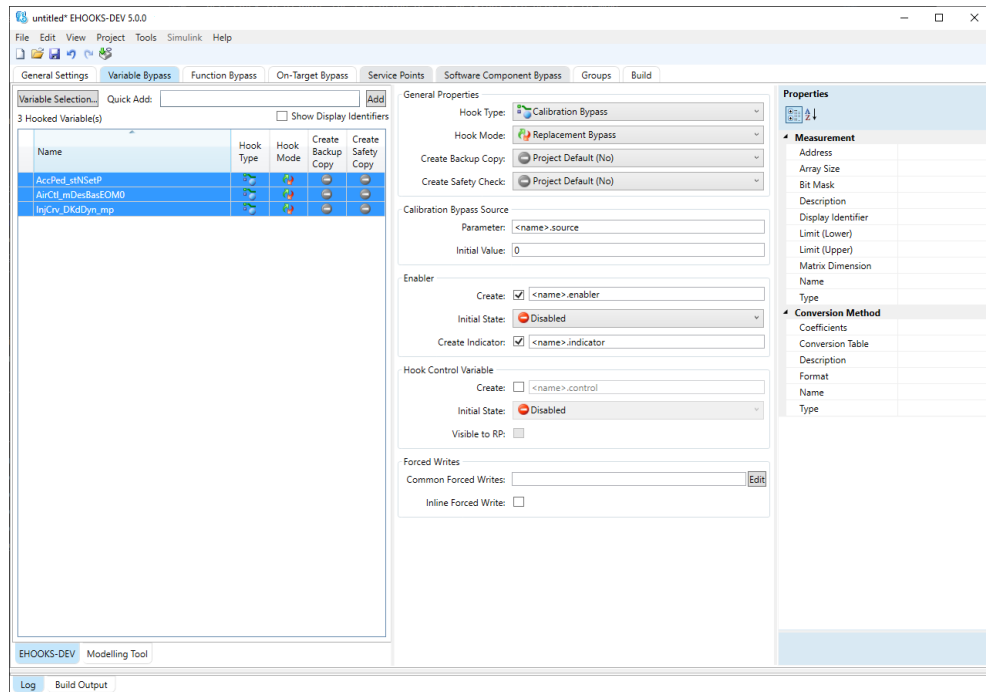


Figure 5.16: Setting Configuration Items for Multiple Hooks

**NOTICE**

*It is possible to edit the forced write location when multiple hooks are selected. In this case, if the selected variables have different forced write locations configured then a dialog will prompt whether the existing configuration should be discarded for all the selected variables and replaced with the new configuration.*

## 5.2.4 Copy and Paste

EHOOKS-DEV supports copy-and-paste functionality that allows hooks to be copied between separate instances of the user interface. This is particularly useful when parts of an existing EHOOKS project configuration are to be reused in another new or modified EHOOKS project.

To use this feature two instances of the EHOOKS-DEV user interface must be open at the same time. The first instance (Project 1) must contain the EHOOKS configuration information to be copied from. The second instance (Project 2) will contain the EHOOKS project to be pasted into, and must use the same input hex and a2l files (defined in the General Settings tab) as Project 1.

From the Variable Selection tab of the Project 1, one or more hooked variables can be copied by selecting those variables in the Hooked variables pane and copying them in the usual way. These hooked variables can then be pasted into Project 2 by clicking in the Hooked variables pane of the Variable bypass tab in the second EHOOKS-DEV instance and pasting in the usual way. Using this method the hooked variables and all of their existing configuration properties from Project 1 are copied into Project 2.

Copy-and-paste functionality is also supported in the Function Bypass, On-target Bypass, Service Points and Group tabs described in the following sections.

### 5.3 Function Bypass Tab

Using the Function Bypass tab, existing ECU functions can be selected to be bypassed and the associated configuration properties can be set. This section details how to set the configuration options; for details on the functionality of each of the configuration items please see section [EHOOKS-DEV Features](#), and for details on how to work with hooked ECU software created with the various configuration options please see section [6 Working with Hooked ECU Software in INCA](#).

Clicking the **Function Selection...** button displays an ECU process selection dialog. In the left-hand column is a list of the ECU processes that can be bypassed (i.e. prevented from executing). In the right hand column is a list of ECU processes that have been configured for bypass. ECU processes can be moved between the two columns using the buttons **>>** and **<<**, or by pressing **Ctrl+right-arrow** and **Ctrl+left-arrow**. Additionally, double clicking on an entry in either list will move it to the other list.

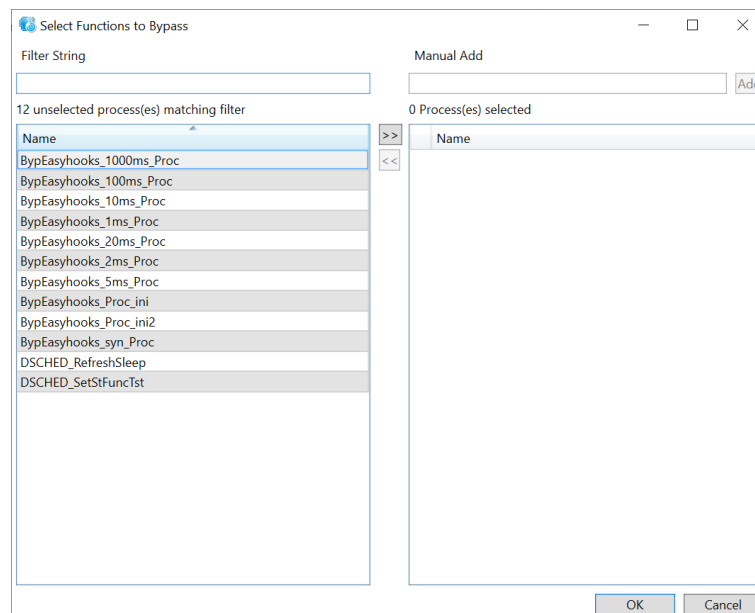


Figure 5.17: Configuring Functions to Bypass

The dialog offers a filter to help locate the desired ECU process; this behaves in the same way as variable selection (see section [Selecting Variables to Be Hooked](#)).

In some situations, the ECU software provider may have prepared some ECU processes to allow them to be bypassed, but indicated that their names should not be displayed in the EHOOKS-DEV user interface. In this case, such functions can be bypassed by typing their name into the **Manual Add** text field and clicking the **Add** button.

The Properties section of the tab allows the Function bypass to be configured. Firstly, the type of the function bypass can be specified as either Dynamic Function Bypass or Static Function Bypass as shown in figure [5.18](#).

Choosing Dynamic Function Bypass from the **Type** drop-down menu allows Enablers, Indicators and Control Variables to be configured for the function bypass. The **Enabler** settings allow the configuration of whether EHOOKS will add a calibration characteristic that can be used at run-time to control whether or not the hook is activated. To configure an enabler, the checkbox must be ticked and then the name of the enabler calibration

The screenshot displays three configuration sections:

- Properties:** A dropdown menu labeled 'Type:' is set to 'Dynamic Function Bypass'.
- Enabler:**
  - 'Create:' checkbox is checked, with the text field containing 'BypEasyhooks\_10ms\_Proc.enabler'.
  - 'Initial State:' dropdown menu is set to 'Disabled'.
  - 'Create Indicator:' checkbox is checked, with the text field containing 'BypEasyhooks\_10ms\_Proc.indicator'.
- Hook Control Variable:**
  - 'Create:' checkbox is checked, with the text field containing 'BypEasyhooks\_10ms\_Proc.control'.
  - 'Initial State:' dropdown menu is set to 'Disabled'.
  - 'Visible to RP:' checkbox is checked.

Figure 5.18: Properties configuration showing Function Bypass type dropdown menu

characteristic can be set in the text field. The initial state of the enabler is set to disabled by default, which causes the function bypass hook to be turned off at ECU startup. The initial state of the enabler can be changed to enabled, which allows the function bypass hook to be turned on at ECU startup.

If an enabler is configured then it is also possible to configure an indicator. The **Indicator** setting allows the configuration of whether EHOOKS will add a measurement variable that will mirror the current value of the enabler characteristic at run-time. To configure an indicator, the checkbox must be ticked and then the name of the indicator measurement variable can be set in the text field.

A hook control variable can also be added to the configuration by ticking the **Create** checkbox in the relevant section and setting the name to be used for the control variable. The initial value of the hook control variable is set to disabled by default, which causes the Dynamic function bypass to be turned off at ECU startup. The initial value of the hook control variable can be changed to enabled, which will cause the Dynamic function bypass to be turned on at ECU startup. The hook control variable can also be configured to allow it to be updated from bypass software running on external rapid prototyping hardware by ticking the **Visible to RP** checkbox.

---

**NOTICE**

*The configured control variable name will be created as a C variable that can then be managed at run-time by an on-target or external bypass function. Therefore the configured name should be a valid C identifier. However, EHOOKS will automatically convert the name into a valid C identifier if the name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (\_ \_). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.*

*The converted C name will need to be used in any on-target or external bypass function that manages a control variable at run-time.*

---

If the bypass type is configured to be Static Function Bypass, then the original function being bypassed can never execute, and it is therefore not possible to configure enablers, indicators

or control variables. When Static Function Bypass is chosen, EHOOKS will re-use the Flash occupied by the original process being bypassed if possible. This can therefore be a useful option for on-target bypass systems where the ECU ROM resources are limited.

---

**NOTICE**

*If the ECU software prepared for EHOOKS by the ECU supplier does not support Function Bypass then the Function Bypass tab will be greyed-out and cannot be used.*

---



---

**NOTICE**

*The re-use of ECU flash resources for Statically bypassed functions is not supported for all ECU types. In this case the Static Bypass option will be greyed out and cannot be used. You should check support for this feature with your ECU supplier.*

---

## 5.4 On-Target Bypass Tab

Using the on-target bypass tab, new software functions can be defined and configured that EHOOKS will integrate into the hooked ECU software. These on-target bypass software functions would typically be used to calculate bypass values for hooked ECU variables, but they can also perform other functions, such as managing hook control variables (see sections [3.2.2 EHOOKS-DEV Hook Configuration Properties - Control variables](#) and [5.2.2 Configuring Properties of a Variable Hook](#)).

---

**NOTICE**

*If the on-target bypass is being configured using ASCET or Simulink, the on-target bypass function cannot be added to the on-target bypass tab manually. ASCET or Simulink should be used to directly add the corresponding hooks and configuration (see section [10 Creating and Working with On-Target Bypass from Simulink](#) or the [ASCET-SE EHOOKS Add-on User Guide](#) for full details – this can be found in the ASCET installation directory within the folder <ASCET\_INSTALL\_DIRECTORY>\target\trg\_ehooks\documents). These on-target bypass functions will appear within the On-Target Bypass tab under the **Modeling Tool** list available at the bottom of the on-target bypass list.*

---

The interface for adding and configuring an on-target bypass function follows a simple three step workflow.

### How to Add and Configure an On-Target Bypass Function

- Step 1: Add an on-target bypass function (see red highlight in figure [5.19](#)). The name must match the name of the C function provided to implement the on-target bypass function. The C file(s) which provide the implementation of the on-target bypass function must be added to the information on the Build tab (see section [5.8 Build Tab](#)). Once added, the on-target bypass function will appear in the list in step 2.
- Step 2: Provides a list of the on-target bypass functions that have been added to the configuration (see blue highlight in figure [5.19](#)). Selecting one from this list allows the associated configuration properties to be set in step 3. To remove an on-target bypass function from the configuration, simply select it and press the Delete key or **right-click -> Delete**. To rename an existing on-target bypass function, first select the row in the

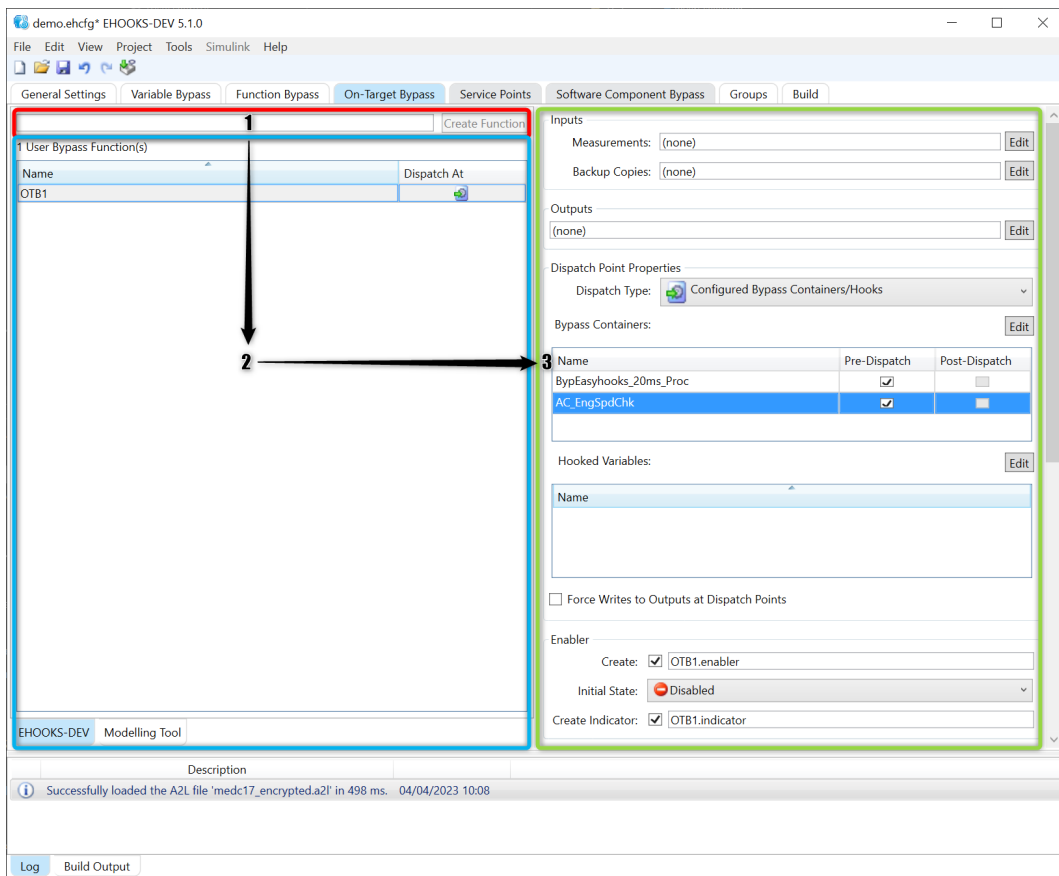


Figure 5.19: On-Target Bypass Configuration

list containing the on-target bypass function and then **click** a second time on the name. The name field within the list grid view will then become editable. To change the properties of an on-target bypass function, simply select it in the list and then proceed with step 3.

- Step 3: Configure the properties to provide the desired on-target bypass configuration for the selected on-target bypass function (see green highlight in figure 5.19).

#### 5.4.1 Configuring Properties of an On-Target Bypass Function

Once an on-target bypass function has been added, it will appear within the list of on-target bypass functions on the On-Target Bypass tab. Selecting an on-target bypass function (or many on-target bypass functions) from within the **on-target bypass** list allows its (their) properties to be configured on the right-hand side.

---

##### NOTICE

*The on-target bypass list is actually two lists that can be viewed independently using the tab control at the bottom of the grid view control. The **EHOOKS-DEV** list contains a list of the on-target bypass functions that have been created and configured within the EHOOKS-DEV tool itself. The **Modeling Tool** list contains a list of the on-target bypass functions that have been created and configured by an external modeling tool (such as ASCET or Simulink). The list of on-target bypass functions and their configurations in the **Modeling Tool** list cannot be changed directly within the EHOOKS-DEV user interface; the EHOOKS configuration environment of the modeling tool should be used to achieve this.*

---

The **Inputs** section allows the configuration of the ECU variables that should be made available as inputs to the on-target bypass function. This consists of two types of inputs:

- **Measurements**

Measurement inputs are ECU measurement variables. The current value of the selected ECU measurement variables will be used as the input to the on-target bypass function. This means that, where a bypass hook has been configured (and is activated) for an ECU variable, the current bypass value will be used as the input value.

- **Backup Copies**

Backup copy inputs allow the original ECU calculations to be used as inputs for ECU variables that have been hooked. The backup copy inputs provide the original ECU calculation as stored in a backup copy. To use this type of input, a backup copy must have been configured for the associated hooked ECU variable (see section [3.2.2 EHOOKS-DEV Hook Configuration Properties](#)).

Both types of inputs are added using the same variable selection interface. Selecting the **Edit** button will display a new variable section dialog, see figure 5.21. The left-hand column allows the displayed variable list to be filtered by the A2L file function groups (including filtering by the input, output and local measurements to a function). To quickly move to the desired function group, click on an entry in the function group list to move focus and then begin typing.

The middle-column displays a variable list which can be filtered by typing a filter string into the text box. The filter string can include the wildcard characters ? and \*. ? will match any single character and \* will match any number of characters. Note that the filter string contains an implicit \* wildcard at the end of the string. The character \$ can be used to match the end

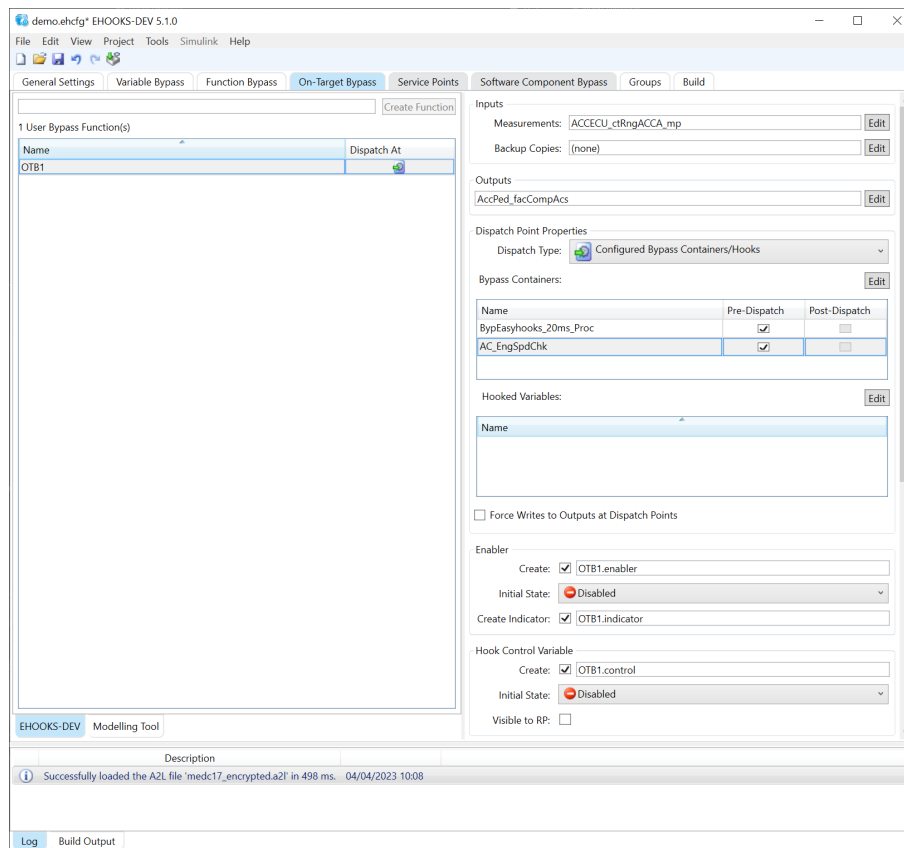


Figure 5.20: Setting On-Target Bypass Function Properties

of a variable name. As the filter string is updated, the variable list will dynamically update to show any matching ECU variables that can be used as inputs to an on-target bypass function.

The right-hand column shows the ECU variables that have been selected as inputs to the on-target bypass function. ECU variables can be moved in and out of this list using the >> and << buttons, or by pressing Ctrl+right-arrow and Ctrl+left-arrow, respectively. Additionally, double clicking on an entry in either list will move it to the other list. If an array variable has been selected as an input to an on-target bypass function, the elements of the array to be accessed can be configured by clicking on the ... button. The number of hooked elements is then shown in the right-hand column.

The **Show Display Identifiers** check box allows the list of variable names to be changed to show the DISPLAY\_IDENTIFIER fields from the A2L file.

The variable selection dialog (figure 5.21) also supports the **Export** and **Import** of lists of variables to be used as inputs to an on-target bypass model. This feature is useful for reusing lists of variables in more than one EHOOKS project. Clicking on the **Export** button brings up a new Windows Explorer dialog that allows the currently selected list of variables to be exported to an EHOOKS-DEV Measurement (\*.mst) file. Clicking the **Import** allows an existing .mst file to be imported. The variables listed in the selected .mst file will then be added to the selected variables list in the variable selection dialog. A warning message will be displayed in the EHOOKS-DEV Log if any of the variables imported from the .mst file are not available for use in the loaded A2L file as an input to an on-target bypass model.

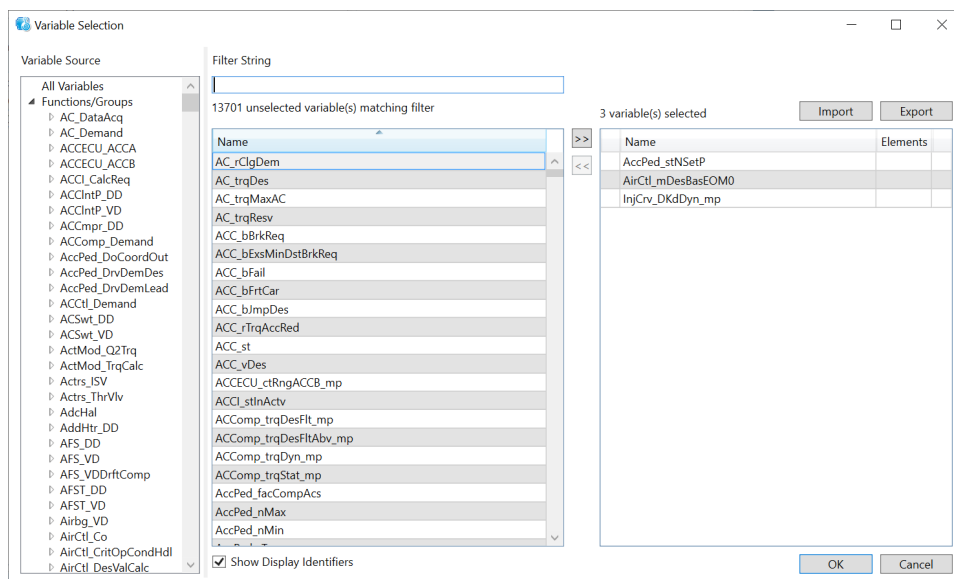


Figure 5.21: Configuring Input Variables for an On-Target Bypass Function

The **Output** section allows the configuration of the ECU variables that should be hooked by EHOOKS and which will have bypass values calculated by the on-target bypass function. The outputs are configured using a variable selection dialog box in the same manner as described above and shown in figure 5.21.

The **Dispatch Point Properties** box allows the configuration of when the on-target bypass function should be executed at run-time. EHOOKS provides two different methods for controlling the run-time execution of an on-target bypass function.

- **Bypass Containers / Hooks**



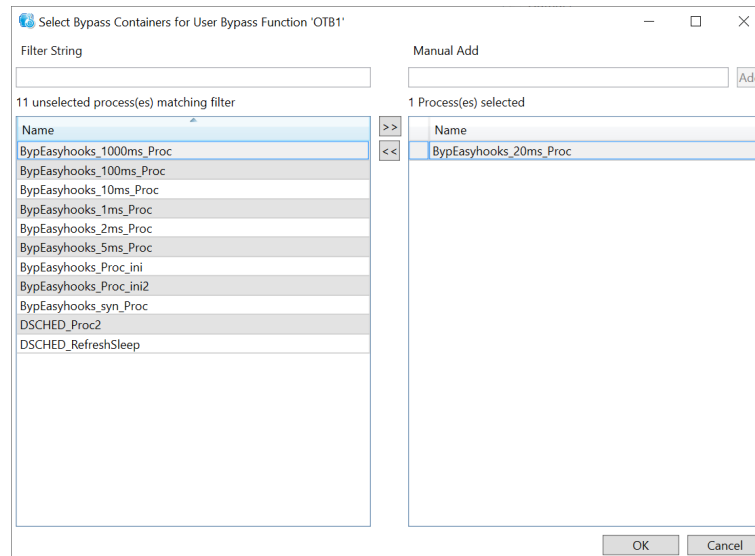


Figure 5.22: Configuring the Dispatching of an On-Target Bypass Function

Dispatching an on-target bypass function within one or more bypass containers and/or one or more hooks allows the configuration of the exact ECU processes or EHOOKS hooks within which the on-target bypass function should be called. Once the **Dispatch Type** is set to **Configured Bypass Containers / Hooks**, the configured bypass containers and hooked variable lists are shown in the read-only **Bypass Containers** and **Hooked Variables** text fields.

The configuration can be changed by clicking one of the **Edit** buttons. This will display the **Select Bypass Containers** or **Select Hook Dispatch Points** dialog as appropriate.

In the **Select Bypass Containers** dialog the left column displays a list of the available ECU processes. EHOOKS can be configured to call the on-target bypass function in one or more of these processes by moving them into the right column by first selecting them and then using the **>>** button. To remove a dispatch process from the configuration, select it and click the **<<** button. The dialog offers a filter to help locate the desired ECU process; this behaves in the same way as variable selection (see section [5.2.1 Selecting Variables to Be Hooked](#)).

In some situations the ECU software provider may have prepared some ECU processes to allow their use as bypass containers, but indicated that their names should not be displayed in the EHOOKS-DEV user interface. In this case, such bypass containers can be added to the configuration by typing their name into the **Manual Add** text field and clicking the **Add** button.

Once the Bypass Containers have been configured as described above, they will then be shown in the Bypass Containers section of the main **On-Target Bypass** Tab. The On-Target bypass function can then be configured to be dispatched before the Bypass Container runs (**Pre-Dispatch**) and/or after the Bypass container has run (**Post-Dispatch**) by using the checkboxes, as shown below.

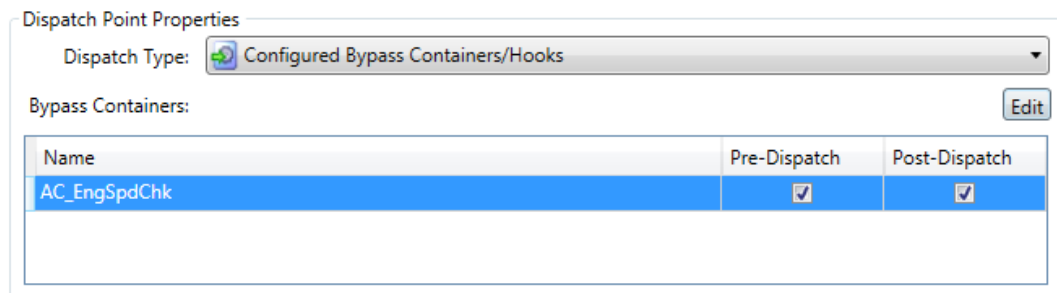


Figure 5.23: Configuring Pre/Post dispatch of the On-Target Bypass Function

**NOTICE**

All Bypass containers can be used to dispatch an On-target bypass function before the Bypass Container (Pre-Dispatch) runs. Some Bypass Containers may also be capable of dispatching the On-target bypass function after the Bypass Container (Post-Dispatch) has run, but this option has to be specifically configured by the ECU software provider as part of their EHOOKS preparation process. If this Post-Dispatch option is required you should therefore discuss this with your ECU supplier.

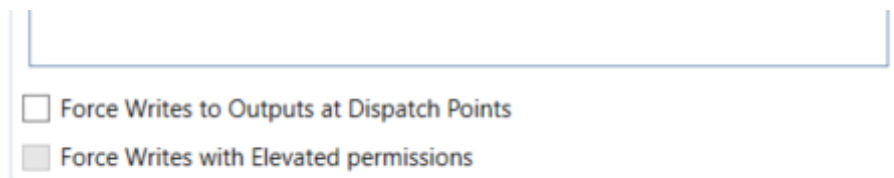


Figure 5.24: On-Target Bypass Force Writes

The **Force Writes to Outputs at Dispatch Points** checkbox enables forced writes of the on-target bypass function's output values, after it has executed.

If the user needs to bypass memory protection, the **Force Writes with Elevated Permissions** option can be selected. The **Force Writes to Outputs at Dispatch Points** option must be enabled first, as well as enabling the advanced project option for elevated permissions (see section [11.5.4 Enable Elevated Permissions](#)). At the dispatch point, forced writes are performed with elevated permissions.

In the **Select Hook Dispatch Points** dialog the left column displays a list of the available EHOOKS hooked ECU measurement variables. EHOOKS can be configured to call the on-target bypass function in one or more of these hooks by moving them into the right column by first selecting them and then using the >> button. To remove a dispatch hook from the configuration, select it and click the << button. The dialog offers a filter to help locate the desired ECU process; this behaves in the same way as variable selection (see section [5.2.1 Selecting Variables to Be Hooked](#)).

- **All Function Output Variables**

Dispatching an on-target bypass function at the variable hook means that, each and every time any of the ECU variables configured as outputs to the on-target bypass function are written within the ECU software, the on-target bypass function will first be executed.

**WARNING**

*Dispatching an on-target bypass function at all function output variable hooks, especially where more than one output variable is configured for the on-target bypass function, means that the function will likely be executed multiple times, potentially in multiple different ECU processes. It is therefore necessary to take additional care and ensure that such a configuration doesn't overload the computational abilities of the ECU. Also, as the on-target bypass function may be called from several different ECU processes which may be able to pre-empt one another, it is also necessary to ensure that the function is implemented in a safe, re-entrant manner.*

A typical use-case for variable dispatch of an on-target bypass function is to implement some simple filtering (or perhaps noise injection) to a single ECU variable. In this case, the same single ECU variable would be configured as both an input and an output of the on-target bypass function. For this specific use-case, where the same single ECU variable is configured as both an input and an output for an on-target bypass function dispatched at the variable hook, it is not necessary read from the backup copy of the hooked variable. The input will always be the current calculation just performed by the original ECU software.

The **Enabler** settings allow the configuration of whether EHOOKS will add a calibration characteristic that can then be used at run-time, to control whether the on-target bypass function is activated or not. If an enabler is configured for an on-target bypass function, it will only ever be executed according to the following logic:

- If a global enabler is configured and is set to disabled, the on-target bypass function will not be executed.
- If a global enabler is configured and is set to enabled, or a global enabler is not configured:
  - If any group enabler associated with the on-target bypass function is enabled, or the on-target bypass function specific enabler is enabled, or the on-target bypass function specific control variable (see below) is enabled, the on-target bypass function is executed.
  - If all group enablers associated with the on-target bypass function are disabled, the on-target bypass function specific enabler is disabled, and the on-target bypass function specific control variable is false, the on-target bypass function is not executed.

To configure an enabler, the checkbox must be ticked and then the name of the enabler calibration characteristic can be set in the text field. The initial state of the enabler is set to disabled by default, which causes the on-target bypass function to be turned off at ECU startup. The initial state of the enabler can be changed to enabled, which will cause the on-target bypass function to be turned on at ECU startup if the above conditions are valid.

If an enabler is configured, it is also possible to configure an indicator. The **Indicator** setting allows the configuration of whether EHOOKS will add a measurement variable that will mirror the current value of the enabler characteristic at run-time. To configure an indicator, the checkbox must be ticked and then the name of the indicator measurement variable can be set in the text field.

A hook control variable can also be created for the On-target bypass function by ticking the

The image shows a configuration window with two main sections: 'Enabler' and 'Hook Control Variable'.  
 In the 'Enabler' section:  
 - 'Create:' is checked, with the text 'NewModel.enabler' in the adjacent field.  
 - 'Initial State:' is a dropdown menu currently showing 'Disabled'.  
 - 'Create Indicator:' is checked, with the text 'NewModel.indicator' in the adjacent field.  
 In the 'Hook Control Variable' section:  
 - 'Create:' is checked, with the text 'NewModel.control' in the adjacent field.  
 - 'Initial State:' is a dropdown menu currently showing 'Disabled'.  
 - 'Visible to RP:' is checked.

Figure 5.25: Configuring On-Target Bypass Function Enabler and Indicator Settings

**Create** checkbox in the relevant section and setting the name to be used for the control variable. The initial value of the hook control variable is set to disabled by default, which causes the On-target bypass function to be turned off at ECU startup. The initial value of the hook control variable can be changed to enabled, which will cause the On-target bypass function to be turned on at ECU startup. The hook control variable can also be configured to allow it to be updated from bypass software running on external rapid prototyping hardware by ticking the **Visible to RP** checkbox.

---

*NOTICE*

*The configured control variable name will be created as a C variable that can then be managed at run-time by an on-target or external bypass function. Therefore the configured name should be a valid C identifier. However, EHOOKS will automatically convert the name into a valid C identifier if the name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (\_ \_). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.*

*The converted C name will need to be used in any on-target or external bypass function that manages a control variable at run-time.*

---

Finally, the **Example Code** section presents an overview of how the on-target bypass function implementation should be structured. This is a read-only display, but it can be **copied** (**right-click Select All** then **right-click Copy**, or select the text and Ctrl+C) and **pasted** (Ctrl+V) into a text editor to form the basis for the on-target bypass function implementation. Once the implementation has been created for each configured on-target bypass function, the associated implementation files (C files, object files and library files) should be added to the **Build** tab so that EHOOKS can include them in the build process (see section [5.8 Build Tab](#)).

On-Target Bypass Example Code

```
/* Include EHOOKS-DEV generated header file */
#include "UserBypassFuncs.h"

EH_USER_BYPASS_FUNC(My0TBFunc)
{
    EH_ARG_PUT_AC_trqDes(EH_ARG_GET_AccPed_stNSetP(EH_context) +
```

```

EH_ARG_GET_AirCtl_mDesBasEOM0(EH_context));

EH_ARG_PUT_InjCrv_qPiI1Des__0__(EH_ARG_GET_AirCtl_mDesBasEOM0(EH_context) -
EH_ARG_GET_AccPed_stNSetP(EH_context));

/* Bypass values valid for use */
return 1;
}

```

## 5.5 Software Component Bypass Tab

For Autosar based ECU software the **Software Component Bypass** tab (see figure ) allows users to configure Runnables belonging to supported Software Components that will be replaced with local versions.

On the left is a list of Software Components that have been configured for bypass. Clicking on one of those shows the configuration options for the bypass in the middle panel including a list of Runnables that belong to the Software Component. Clicking on one of the listed Runnables shows, via the panel on the right, the Ports and Inter-Runnable-Variables that the runnable (and therefore the Runnable's bypass function) has access to.

ARXML files and new C source code implementations of Runnables may be added to the build via the Source Files selector under the Build tab (see section [5.8.1 Configuring Build Source Files](#).)

Configuration may be done in a modelling tool such as Ascet or Simulink, in which case bypassed Software Components are listed via the **Modelling Tool** lower tab.

### 5.5.1 Selecting Software Components for Bypass

Clicking on **Software Component Selection** at the top left of the Software Component Bypass Tab (figure ) displays a list of all supported Autosar Software Components on the left and all selected Software Components on the right.

The Autosar short-name of the software component is listed. Items may be added or removed from the list of Software Component Bypasses by selecting the Software Component short-name and clicking on >> or <<. **OK** confirms the changes.

### 5.5.2 Software Component Bypass Configuration

Clicking on one of the bypassed Software Components shows the configuration options for the bypass in the middle panel (see figure ).

The **Type** may be **Dynamic** which means that control variables (enablers) are used at run-time to determine whether the bypass runnable runs (enabled) or whether the runnable in the original ECU software runs (disabled).

In addition to the global EHOOKS enabler an enabler and indicator may be created via the **Enabler** panel. There the enabler and indicator can be renamed and the initial state of the enabler can be configured.

The **Type** may be **Static** which means that the selected Runnables are permanently bypassed in the hooked ECU. In this case there are no control variables (enablers) and the global RAM space associated with the permanently disabled Runnables in the original ECU software is made available for reuse.

The lower section of the middle panel shows a list of Runnable (short-names) that belong to the Software Component. Clicking on one of the listed Runnables shows, via the panel on the

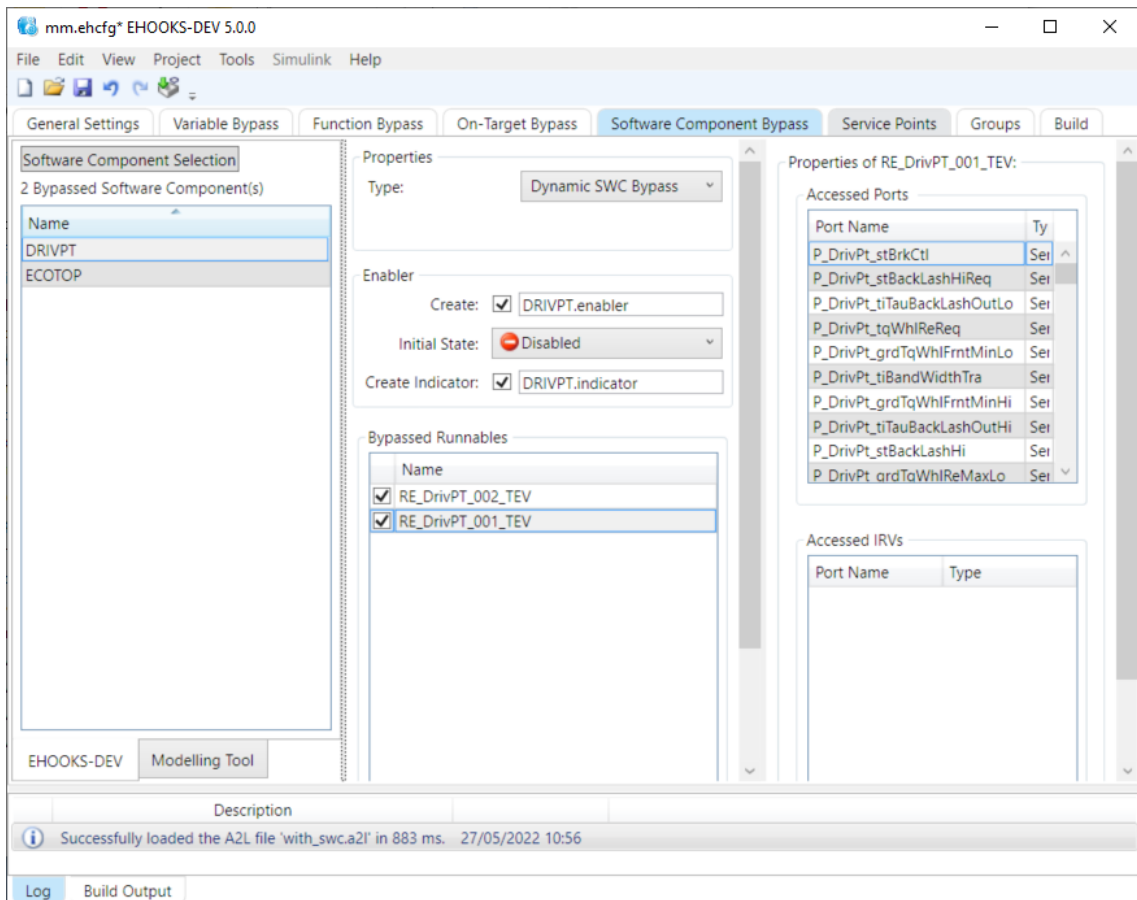


Figure 5.26: The Software Component Bypass tab.

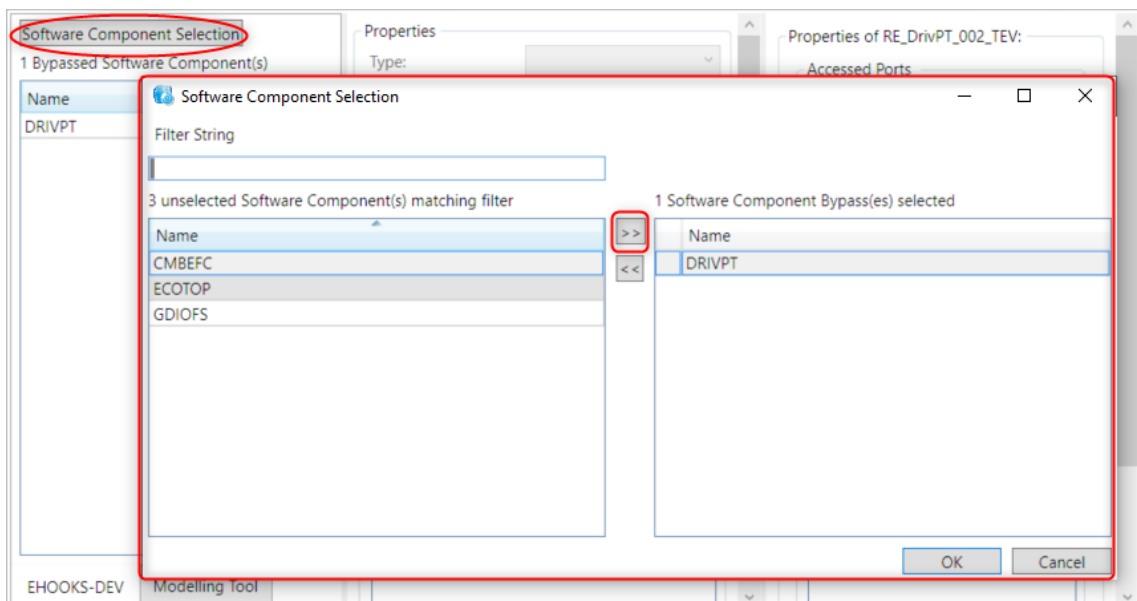


Figure 5.27: Selection of Software Components for Bypass.

right, the Ports and Inter-Runnable-Variables that the runnable (and therefore the Runnable's bypass function) has access to. A check-box next to the runnable short-name allows the Runnable to be selected for bypass. By default all Runnables are selected for bypass when the Software Component is added to the EHOOKS list of Bypassed Software Components.

When the Software Component Bypass has been configured with an Enabler then each bypassed runnable also has an enabler the name of which will be <runnable-name>.enabler. At run-time a Runnable is bypassed when either its Software Component enabler or its Runnable enabler is true:

```
if ((ComponentA__enabler) || (RE_ComponentA_Runnable1__enabler))
{
    ...
    G270_A_Runnable1();
}

if ((ComponentA__enabler) || (RE_ComponentA_Runnable2__enabler))
{
    ...
    G270_A_Runnable2();
}
```

Runnable enablers are therefore initially disabled allowing the Software Component enabler to control at run-time whether all of the Runnables in a Software Component use the original ECU code (false) or the bypass code (true).

To control runnables individually, the Software Component enabler should be set to false and then the individual runnable enablers can be used to control whether to use the original runnable code or the bypass runnable code.

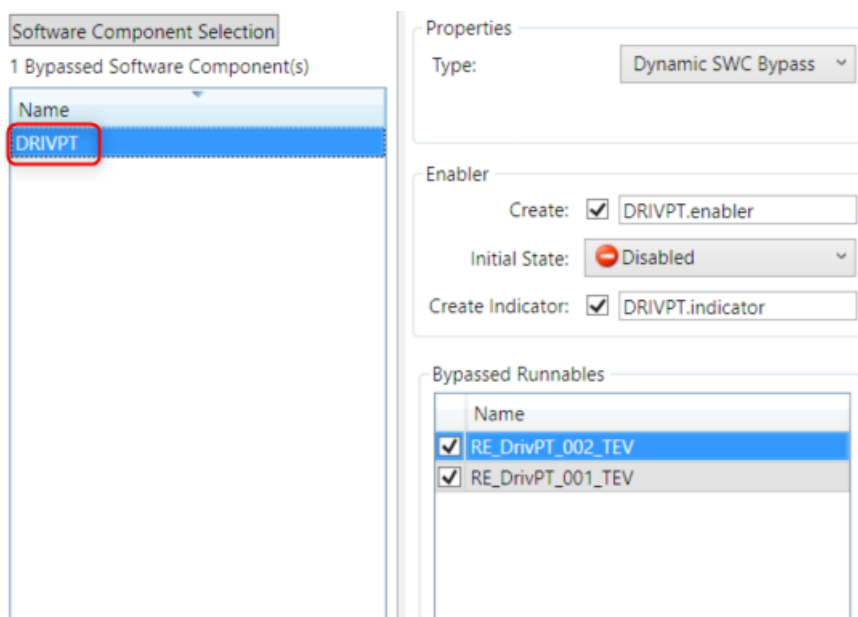


Figure 5.28: Bypassed Software Component detail.

## 5.6 Service Points Tab

Using the Service Points tab shown in figure 5.29, processes to be bypassed by service points can be selected and their associated configuration properties can be set.

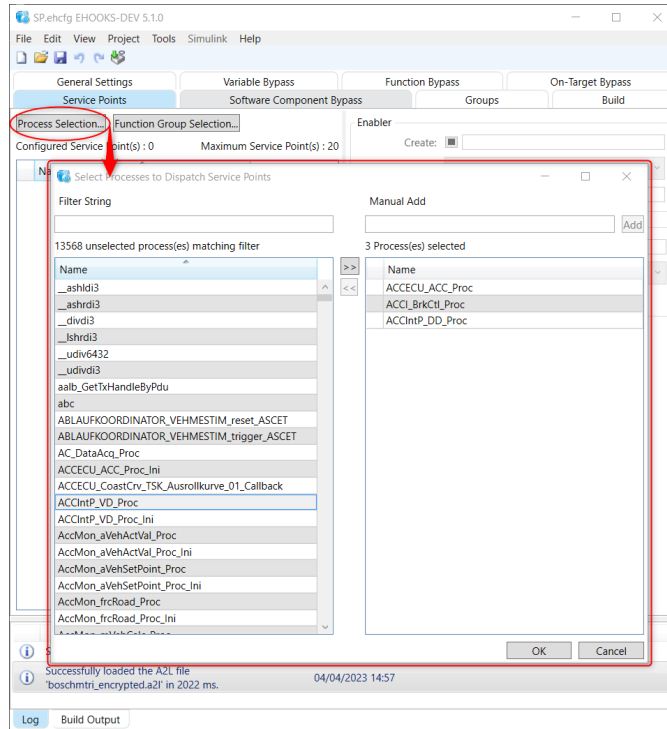


Figure 5.29: The Service Points tab showing the Process Selection dialog

Clicking the **Process Selection...** button from within the Service Points tab will display a process selection dialog. In the left-hand column is a list of the ECU processes that can be bypassed by a service point. These processes are specified by the ECU supplier during their EHOOKS preparation process. In the right hand column is a list of ECU processes that have been configured to be bypassed by service points. ECU processes can be moved between the two columns using the buttons **>>** and **<<**, or by pressing **Ctrl+right-arrow** and **Ctrl+left-arrow**. Additionally, double clicking on an entry in either list will move it to the other list.

The dialog offers a filter to help locate the desired ECU process; this behaves in the same way as variable selection (see section [Selecting Variables to Be Hooked](#)).

In some situations, the ECU software provider may have prepared some ECU processes to allow them to be bypassed as service points, but indicated that their names should not be displayed in the EHOOKS-DEV user interface. In this case, such functions can be bypassed by typing their name into the **Manual Add** text field and clicking the **Add** button.

Service points can also be added by clicking the **Function Group Selection...** button from within the Service Points dialog. This provides a faster way to add service points for all processes within a function. The **Function Group Selection...** dialog works in a similar way to the **Process Selection...** dialog described above (but without the **Manual Add** text field). When a Function Group is selected a service point will be added for every process in that Function Group. If the group is called Func1 then the service points will be named Func1\_1, Func1\_2, Func1\_2....

The total number of configured service points is indicated in the Service Points tab. Also



indicated is the maximum number of service points that is allowed to be configured, as defined by the ECU software supplier during the preparation of the EHOOKS-ready ECU software. In the event that the total number of configured service points exceeds the maximum allowed number of service points, an error icon is displayed to inform the user. If this situation occurs, the number of configured service points must be reduced to be less than or equal to the maximum allowed number.

---

**NOTICE**

*If the ECU software prepared for EHOOKS by the ECU supplier does not support Service Based Bypass then the Service Points tab will be greyed-out and cannot be used.*

---

The **Enabler** settings in the Service Points tab allow the configuration of whether EHOOKS will create a new calibration characteristic that can be used at run-time to control whether or not the Service Point is activated. To create an enabler, the **Create** checkbox must be ticked and then the name of the enabler characteristic can be set in the text field. The initial state of the enabler is set to disabled by default, which causes the Service point to be turned off at ECU startup. The initial state of the enabler can be changed to enabled, which allows the Service point to be turned on at ECU startup.

If an enabler is configured then it is also possible to configure an indicator. The **Indicator** setting allows the configuration of whether EHOOKS will add a measurement variable that will mirror the current value of the enabler characteristic at run-time. To configure an indicator, the checkbox must be ticked and then the name of the indicator measurement variable can be set in the text field.

The screenshot displays two configuration sections. The first section, titled 'Enabler', contains three rows: 'Create:' with a checked checkbox and the text 'EHOOKS\_IMPL\_process\_R02B.enabler'; 'Initial State:' with a dropdown menu showing 'Disabled' and a red minus icon; and 'Create Indicator:' with a checked checkbox and the text 'EHOOKS\_IMPL\_process\_R02B.indicator'. The second section, titled 'Hook Control Variable', contains three rows: 'Create:' with a checked checkbox and the text 'EHOOKS\_IMPL\_process\_R02B.control'; 'Initial State:' with a dropdown menu showing 'Disabled' and a red minus icon; and 'Visible to RP:' with a checked checkbox.

Figure 5.30: Configuring Service Point properties

A hook control variable can also be added to the configuration by ticking the **Create** checkbox in the relevant section and setting the name to be used for the control variable. The initial value of the hook control variable is set to disabled by default, which causes the Service point to be turned off at ECU startup. The initial value of the hook control variable can be changed to enabled, which will cause the Service point to be turned on at ECU startup. The hook control variable can also be configured to allow it to be updated from bypass software running on external rapid prototyping hardware by ticking the **Visible to RP** checkbox.

**NOTICE**

The configured control variable name will be created as a C variable that can then be managed at run-time by an on-target or external bypass function. Therefore the configured name should be a valid C identifier. However, EHOOKS will automatically convert the name into a valid C identifier if the name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (`__`). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.

The converted C name will need to be used in any on-target or external bypass function that manages a control variable at run-time.

For a function group 'Func1' an individual enabler, indicator and control variable will be created for each service point `Func1_1`, `Func1_2`, `Func1_2...` within the group when the checkboxes shown in figure 5.30 are ticked.

The initial state of the enabler is set to disabled by default, which causes the Service Point to be disabled at ECU startup. The initial state of the enabler can be changed to enabled, which will cause the Service Point to be enabled at ECU startup according to the following conditions:

- If a global enabler is configured and is set to disabled, then all Service Points are **disabled**.
- If a global enabler is configured and is set to enabled, or a global enabler is not configured; then
  - If any group enabler associated with the Service Point is enabled, or the Service Point specific enabler is enabled, or the Service Point specific control variable (see section 3.2.2.3 Control Variables) is enabled, then the Service Point is **enabled**.
  - If all group enablers associated with the Service Point are disabled and the Service Point specific enabler is disabled and the Service Point specific control variable is false, then the Service Point is **disabled**.

If a service point is disabled then the service point does not run and the original ECU process runs as normal. If a service point is enabled then the behavior of the service point is controlled via INTECRIO and the standard Service Based Bypass control characteristics provided by the ECU supplier. Please refer to the INTECRIO user guide for more information.

## 5.7 Group Tab

Using the group tab allows hook groups to be defined and configured. Hook groups allow entire groups of hooks to be enabled/disabled with a single group enabler characteristic (according to the logic stated in [Hook Enablers](#)), and monitored with a single group indicator measurement. The interface for adding and configuring groups follows a simple three step workflow.

How to Add and Configure a Group

- Step 1: Add a group (see red highlight in figure 5.31) by specifying a group name and click **Create Group**.
- Step 2: Provides a list of the configured groups (see yellow highlight in figure 5.31). To

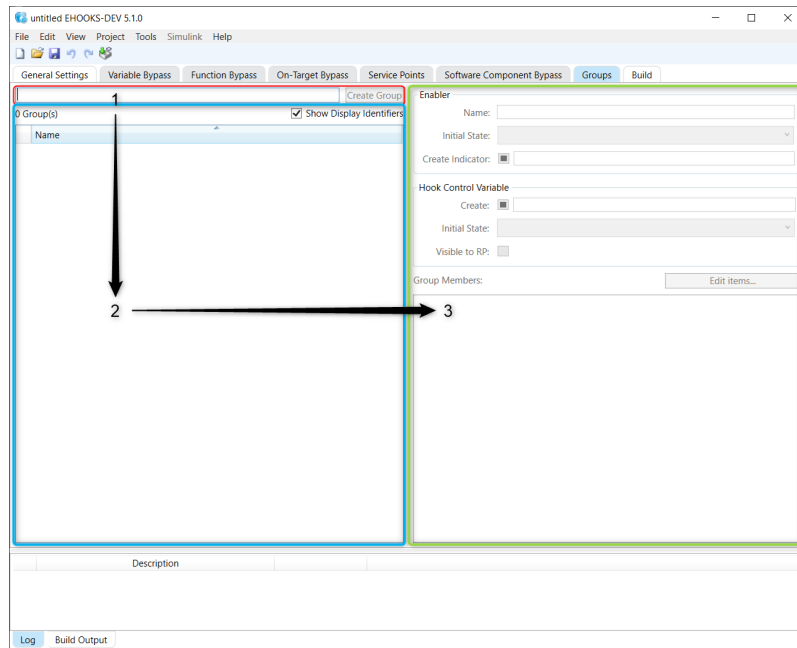


Figure 5.31: Group Configuration

remove a group from the configuration simply select it and press the Delete key or **right-click -> Delete**. To rename an existing group, first select the row in the list containing the group and then **click** a second time on the name. The name field within the list view will then become editable. To change the properties of a group, simply select it in the list and then proceed with step 3.

- Step 3: Configure the properties to provide the desired group configuration for the selected group (see green highlight in figure 5.31).

#### 5.7.1 Configuring the properties of a Group

The **Enabler** settings allow the configuration of the enabler calibration characteristic that EHOOKS will create, which can be used at run-time to control whether the hook group is activated or not. The initial value of the group enabler is set to disabled by default, which causes the group to be deactivated at ECU startup. The initial value of the group enabler can be changed to enabled, which will cause the group to be activated at ECU startup.

The **Indicator** setting allows the configuration of whether EHOOKS will add a measurement variable that will mirror the current value of the group enabler characteristic at run-time. To configure an indicator, the checkbox must be ticked and then the name of the indicator measurement variable can be set in the text field.

A Group **Hook control variable** can also be added to the configuration by ticking the **Create** checkbox in the relevant section and setting the name to be used for the control variable. The **Initial State** of the Group hook control variable is set to Disabled by default, which causes the hook to be turned off at ECU startup. The **Initial State** of the hook control variable can be changed to Enabled, which will cause the hook to be turned on at ECU startup. The hook control variable can also be configured to allow it to be updated from bypass software running on external rapid prototyping hardware by ticking the **Visible to RP** checkbox.

**NOTICE**

The configured control variable name will be created as a C variable that can then be managed at run-time by an on-target or external bypass function. Therefore the configured name should be a valid C identifier. However, EHOOKS will automatically convert the name into a valid C identifier if the name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (\_\_). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.

The converted C name will need to be used in any on-target or external bypass function that manages a control variable at run-time.

The screenshot shows a configuration dialog with two main sections: 'Enabler' and 'Hook Control Variable'.  
 In the 'Enabler' section:  
 - 'Name:' is set to 'func.enabler'.  
 - 'Initial State:' is a dropdown menu set to 'Disabled'.  
 - 'Create Indicator:' is checked, with the value 'func.indicator' entered.  
 In the 'Hook Control Variable' section:  
 - 'Create:' is checked, with the value 'func.control' entered.  
 - 'Initial State:' is a dropdown menu set to 'Disabled'.  
 - 'Visible to RP:' is checked.

Figure 5.32: Configuring Group Enablers, Indicators and Control Variables

In the Group Members section, it is possible to display the currently configured status of group membership and to edit this configuration. To edit the configuration click **Edit items...** to open the group membership dialog. The group membership dialog has two columns: the left hand column displays all the possible items that can be included in a group and the right hand column displays the items that are included within the current group. Items can be moved between the two columns using the >> and << buttons, by pressing Ctrl+right-arrow and Ctrl+left-arrow, or by double-clicking on entries.

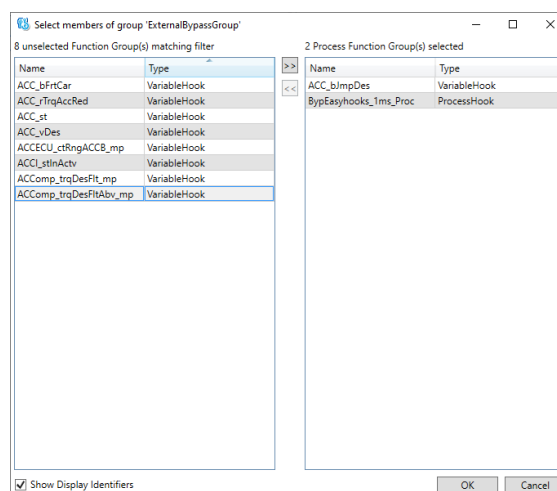


Figure 5.33: Configuring Group Membership

## 5.8 Build Tab

The Build tab allows the EHOOKS build configuration to be customized. The five sub-tabs in the main Build tab allow the configuration of Source Files, Memory Sections, Build Scripts, Build Macro Definitions and Characteristics Groups, as described in the following sections.

### 5.8.1 Configuring Build Source Files

The **Source Files** tab allows the specification of include directories and source files that EHOOKS should use during the build process.

#### NOTICE

*The Source Files tab items are only relevant for the configuration of on-target bypass. If the EHOOKS configuration doesn't make use of on-target bypass, it is not necessary to configure anything in this tab.*

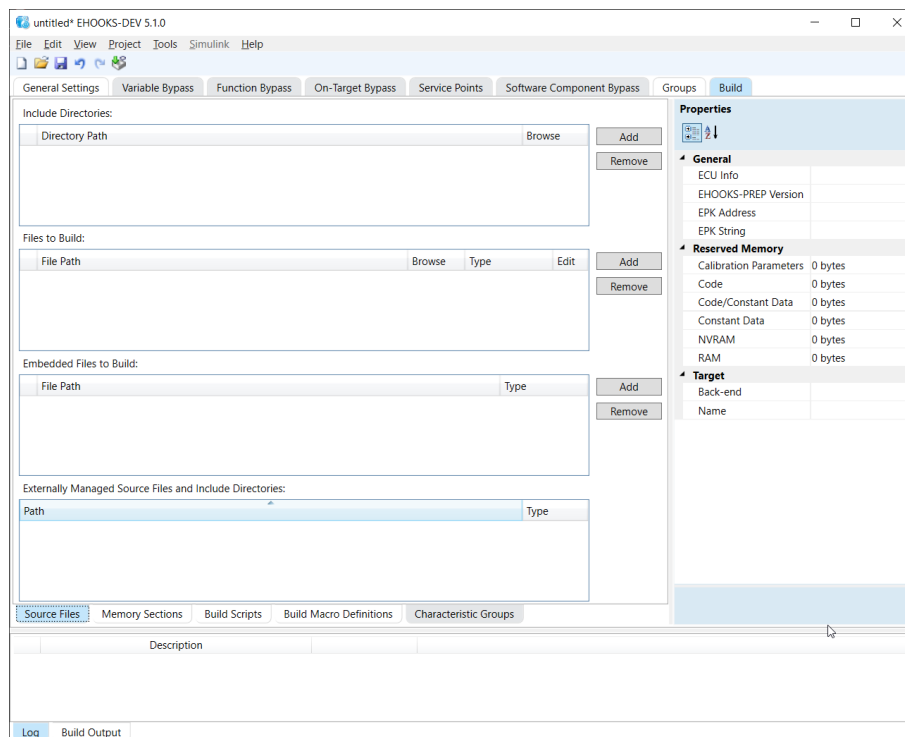


Figure 5.34: Build Configuration

The **Source Files** tab consists of four sections:

- **Include Directories**

Specifies one or more folders that EHOOKS will add to the compiler search path when compiling on-target bypass functions. To add a folder to the include path, click the **Add** button. This will add a new row to the table. Within the row, the folder can be configured by either clicking the **...** button in the **Browse** column and using the resulting folder browser or clicking on path text within the **Directory Path** column and typing in the folder path. The include path can also be quickly added by dragging and dropping from Windows explorer. To remove a folder, first click on the corresponding row and then click the **Remove** button.

- **Files to Build**

Specifies the files that should be used by EHOOKS when building the new hooked ECU software. To add a file to the build list, click the **Add** button. This will add a new row to the table. Once a new row is added, the file path should be edited to locate the associated build file. This can be done either clicking the **...** button in the **File Path** column and using the resulting file browser or clicking on path text within the **File Path** column and typing in the folder path. The files to be built can also be quickly added by dragging and dropping from Windows Explorer.

Once the file path has been configured, the file type should be set. EHOOKS supports the following four file types:

- **C Source File**

This type should be used for C source code files that need to be compiled by EHOOKS as part of the hooked ECU software build process.

- **Object File**

This type should be used for object code files that have already been compiled and are available as .o files that EHOOKS simply needs to link with during the hooked ECU software build process.

- **Library File**

This type should be used for library files that are available as .lib or .a files that EHOOKS simply needs to link against during the hooked ECU software build process.

- **User Definitions File**

This type should be used for EHOOKS User Definition files. These files allow the specification of new measurements and calibrations to support the new on-target bypass algorithm. In addition, one can specify that existing ECU calibration data should be made available for reuse within an on-target bypass algorithm, and define new conversion formulae (COMPU\_METHODS). For user definition files, EHOOKS offers a built-in editor. This can be launched by clicking the **Edit** button within the **Edit** column of the **Files to Build** section. For more details see sections [9 Creating and Working with On-Target Bypass](#) and [11.7.2 EHOOKS-DEV User Definition Files](#).

- **DCM File**

This type should be used for DCM (data calibration) files that are available as .dcm files. EHOOKS will merge the contents of the DCM file into the hooked ECU software during the build process.

- **ARXML File**

ARXML files describe the bypass copy of any Autosar Software Component that is hooked.

- **HEX/S-record File**

This type should be used for hex/s-record files. EHOOKS will merge the contents of the hex/s-record file into the hooked ECU software during the build process. By default, EHOOKS will only merge data contained within data memory sections, however the --merge-all command line option can be used to merge all sections

(code, const and data) of hex/s19 files into the build process.

---

**NOTICE**

*By default, EHOOKS will only merge data sections when merging hex files. The command line option `--merge-all` can be used to override this default behaviour and allow merging of data, code and constant sections.*

---



---

**NOTICE**

*EHOOKS performs consistency checking when merging hex and s-record files as follows:*

*If the A2L has no EPK specified no checking will be done.*

*If the original A2L file includes an EPK, any file merged using the Hex/s-record merge feature will be checked for the correct EPK. If the EPK in the merged file is incorrect, the build will fail with an error.*

*If the original A2L file includes an EPK and if the merged file does not contain any data in the EPK region, the default behaviour is for the build to fail with an error, however the `'-consistency-warn'` option may be used to override the error and allow the build to continue.*

---

- **Embedded Files to Build**

Similar to **Files to Build** enter the name of any files that are embedded in the EHOOKS A2L file and that should be included in the build process. Only filenames (not paths) need to be specified.

- **Externally Managed Source Files & Include Directories**

If on-target bypass is being configured from an external tool (such as ASCET or Simulink), this list will display a read-only list of the files associated with the external configuration.

---

**NOTICE**

*The availability of the read-only externally managed file source file & include directory list depends on the implementation of the external tool and when it synchronizes data with EHOOKS. It is therefore possible that no data will be displayed. This does not imply that the configuration is not being managed correctly.*

---

## 5.8.2 Configuring Memory Sections

The **Memory Sections** tab allows the EHOOKS memory sections defined by the ECU software provider to be updated or overridden.

**DANGER**

When changing the provided memory section information, additional caution must be taken as incorrect specification can easily corrupt the memory of the ECU leading to run-time failures. No changes should be made without reference to the ECU software provider to ensure that the specified memory sections are available, unused in the ECU software and valid for use.

Typically there is no need to change the provided memory section information. However, if the EHOOKS build fails due to insufficient space for code, constant data, calibration parameters or RAM, then it may be possible to update the memory section information to provide more space for EHOOKS to use. For reference, the ECU software providers configured memory sections are displayed in the **Tier-1 Memory Section** area of the dialog.

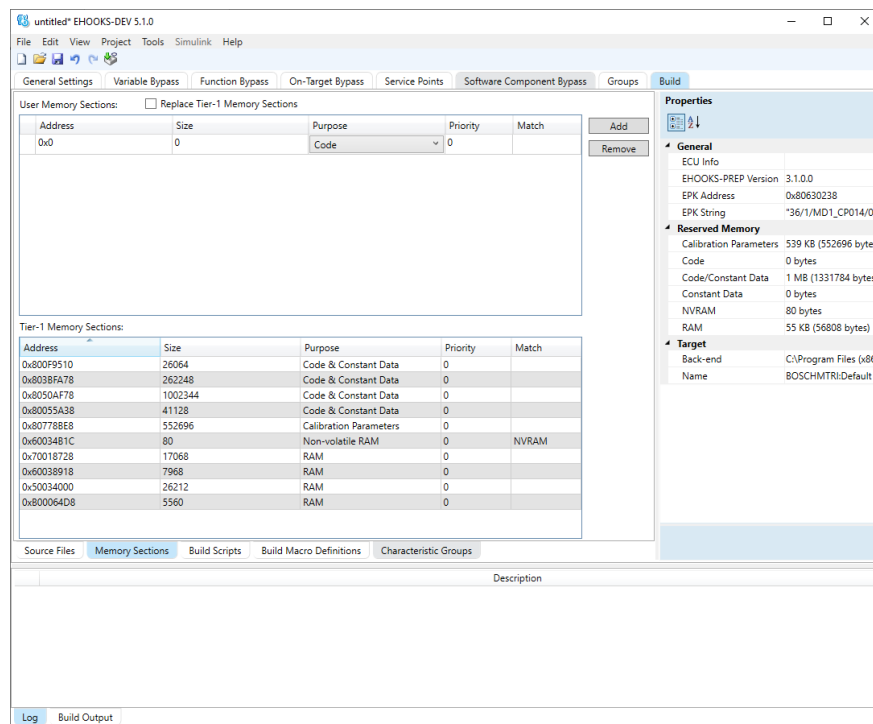


Figure 5.35: Configuring EHOOKS Memory Sections

The new memory section information can be configured by clicking the **Add** button. Then the start address, size and purpose should be defined. The **Purpose** indicates to EHOOKS that the memory section should be used for either code, constant data, code/constant data, calibration parameters, or as RAM or Non-Volatile RAM. EHOOKS will display a warning message if user-defined memory sections overlap with each other or if the HEX file contains data in the range of a user-defined memory section.

**NOTICE**

For Calibration Parameters, only one single memory section is allowed. However, for Code, Constant Data, RAM and Non-Volatile RAM it is possible to add as many memory sections as necessary, including specifying several memory sections for the same purpose.



**Priority** indicates the priority of each memory section. EHOOKS will use memory sections with the highest priority first. Memory section priority can be changed using the editor shown in figure 5.30. The default priority is setting is 0, with positive numbers indicating higher priority and negative numbers indicating lower priorities. Where possible, EHOOKS will try to divide the usage across the defined memory sections.

A memory section can be removed simply by selecting it within the table and clicking the **Remove** button.

The memory sections finally used by EHOOKS will be determined by the following rules:

- If **Replace Tier-1 Memory Sections** is checked, all memory sections displayed in the Tier-1 memory sections list will be ignored and only the user defined memory sections will be used.
- If **Replace Tier-1 Memory Sections** is unchecked:
  - All memory sections displayed in the user memory sections list will be used.
  - Any Tier-1 memory sections that don't overlap in any way with any user memory sections will be used.
  - If an address in a user defined memory section overlaps in any way (even by just 1 byte) with a Tier-1 memory section, that Tier-1 memory section will be ignored.

The **Match** field allows the EHOOKS-DEV user to assign specific memory sections to be used as Special Purpose or Non-Volatile areas of RAM for example. When a regular expression (for example 'SPRAM.\*' or 'NVRAM.\*') is defined in the **Match** field for a specific memory section, any variable name that matches that regular expression will then be assigned to the corresponding section of memory. For example, variables named like NVRAM\_Var1, NVRAM\_Var2, NVRAM\_Var3 in on-target bypass code would be placed into a Non-Volatile RAM section if the regular expression 'NVRAM.\*' is defined in the **Match** field. The benefit is that when running on-target bypass experiments, the value of an EHOOKS-created variables sometimes have to be retained when the ECU is powered off. In such cases the **Match** feature can be used to explicitly place the variable into an area of non-volatile RAM.

### 5.8.3 Configuring pre- and post-build scripting

EHOOKS-DEV allows for custom build steps to be inserted into the EHOOKS build process by supporting the execution of Ruby scripts immediately before and after the hooked ECU files are built. There are six possible types of Ruby scripts that may be executed in the following order:

- tier1\_prebuild.rb
- prebuild\_global.rb
- <project\_prebuild>
- Build of hooked ECU files.
- <project\_postbuild>
- postbuild\_global.rb
- tier1\_postbuild.rb

The tier1\_prebuild.rb and tier1\_postbuild.rb scripts are embedded in the A2L file by the Tier1

as part of their EHOOKS ECU SW preparation process. The EHOOKS-DEV user has no control over these scripts.

The `prebuild_global.rb` and `postbuild_global.rb` scripts are contained in the `<install-dir>\Build` directory. These scripts are used for every build. These scripts may be used to run scripts that should run for every EHOOKS-DEV project.

The `<project prebuild>` and `<project postbuild>` are project-specific scripts that can be specified in the **Build Scripts** tab, as shown in figure 5.36.

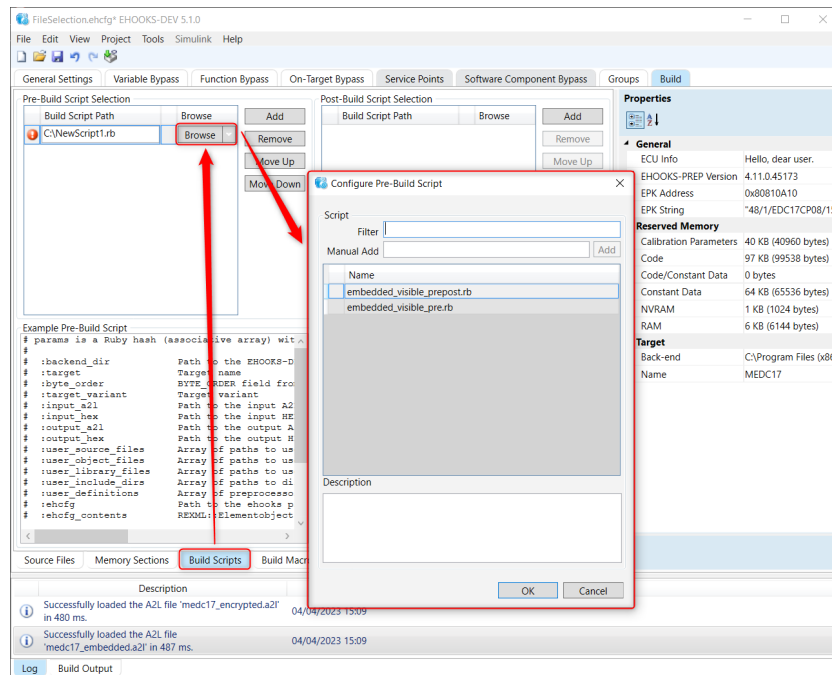


Figure 5.36: Build tab showing the pre- and post- build script configuration dialog

The **Build Scripts** tab includes separate sections for project specific Pre-build and Post-build script selection. Clicking one of the **Add** buttons will allow the addition of a project-specific pre-build or post-build script. Once the **Add** button has been clicked, the selection of a project-specific pre-build or post-build script can be done in one of two ways. Firstly, clicking the **Browse** button allows any project specific build script file to be selected. Alternatively, clicking on the drop-down arrow next to each browse button gives the option to 'Choose Embedded'. When this option is selected, the dialog shown in figure 5.36 allows a script to be chosen from a list of files that have been embedded in the A2L file by the ECU software provider.

If additional project-specific pre-build and/or post-build scripts are required the **Add** button can be clicked again and the selection process repeated. If multiple scripts are configured, the scripts will run in the order shown in the dialog. The **Move Up** and **Move Down** buttons can be used to change the execution order as required.

Pre-build and post-build scripts are passed a Ruby hash as an argument. This contains information about the project. See the `prebuild_global.rb` and `postbuild_global.rb` scripts for information about the contents of the hashes.

**Example:** Assume that you want to run a tool called `chksumgen.exe` to update a checksum in a hooked `.hex/.s19` file. This could be done in either `global_postbuild.rb` or a project

specific postbuild script. For example:

```
def postbuild(params)
  output_hex_file = params[:output_hex]
  chksumgen = "d:\\temp\\chksumgen.exe"
  puts "Postbuild script: #{chksumgen} #{output_hex_file}"
  system( "#{chksumgen} #{output_hex_file}" )
end
```

EHOOKS will apply the following naming convention to messages emitted by scripts:

- If the message text in the script is prefixed by the "ERROR:" tag, the emitted message will also be prefixed by "ERROR:"
- If the message text in the script is prefixed by the "WARNING:" tag, the emitted message will also be prefixed by "WARNING:"
- If the message text in the script does not contain either a "WARNING:" or an "ERROR:" tag, an information message will be emitted by EHOOKS

#### 5.8.4 Configuring macro definitions

EHOOKS allows for macro definitions to be added to the build process using the **Build Macro Definitions** tab, accessed from the main **Build** tab as shown in figure 5.37.

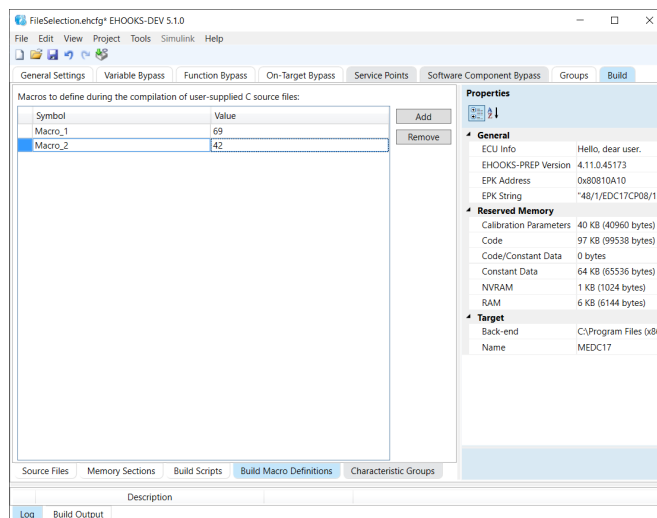


Figure 5.37: Build Macro Definitions Tab

To add new macros to the EHOOKS build click the **Add** button and then type in the symbol names for all macros that are to be added. Values can optionally be added for any of the defined macros.

#### 5.8.5 Configuring Characteristic Groups

Having multiple characteristic groups can be useful if the total number of characteristics created by EHOOKS is too large to be calibrated in a single step, due to RAM restrictions for example. In this case it can be useful to split the characteristics into groups and then calibrate each group separately.

On ECU targets for which multiple characteristic groups are supported, EHOOKS-DEV allows the user to configure a mapping of EHOOKS generated characteristics to specific named

groups. This can be done using the **Characteristic Groups** tab shown in figure 5.38.

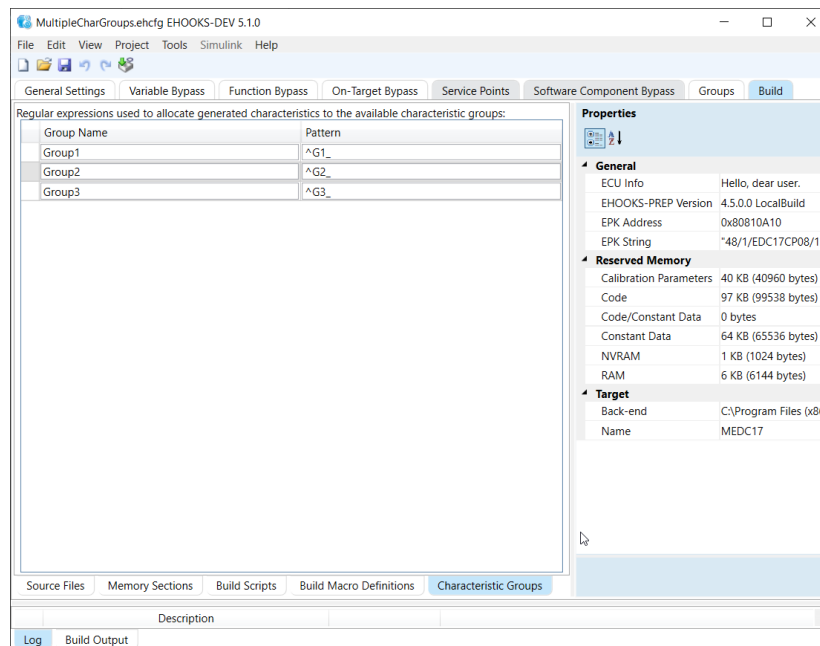


Figure 5.38: Characteristic Groups tab

If the ECU software supplier has prepared their ECU software to have multiple characteristic groups, the Characteristic Groups tab will be active. Otherwise, this tab will be greyed out.

In the example shown in figure 5.38, the ECU software has been prepared with three characteristic groups, and so three lines are available for the user to enter their characteristic group information. In the Group Name column, the user can enter up to three Group Names. For each of these groups, the user must also enter a corresponding regular expression in the Pattern column. When EHOOKS creates a new characteristic with a name that matches one of these regular expressions, the characteristic will be placed into the corresponding group.

## 5.9 Configuration Consistency Checking, Building and Options

### 5.9.1 Consistency Checking

When using the EHOOKS-DEV configuration tool, it continuously monitors the configuration for consistency and any errors. If errors are detected these will be indicated by an error indicator next to the relevant configuration item. See figure 5.39.



Figure 5.39: Error Indicators

Placing the mouse pointer over an error indicator will display a popup tool-tip with details about the source of the error and how it can be corrected, as shown in figure 5.40.

The EHOOKS-DEV configuration tool can automatically correct inconsistencies in the configuration of on-target bypass hooks. These features can be accessed from the **Project On-Target Bypass** menu as shown in figure 5.41.

- Remove inconsistent outputs

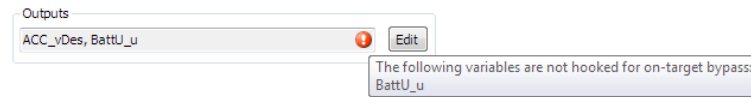


Figure 5.40: Error Indicator Details

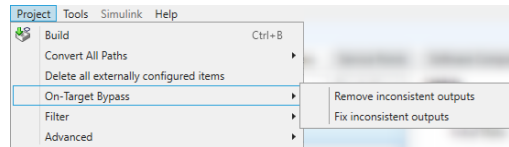


Figure 5.41: On-Target Bypass Automatic Consistency Correction

Any ECU variables added as outputs for an on-target bypass function that are already hooked with a hook type other than on-target bypass will be removed from the on-target bypass configuration.

- Fix inconsistent outputs

Any ECU variables added as outputs for an on-target bypass function that are already hooked with a hook type other than on-target bypass will have their hook type changed to on-target bypass.

### 5.9.2 Building Hooked ECU Software

Once the EHOOKS configuration has been completed, the hooked ECU software can be built using the Project Build menu command or by clicking the **build** button on the tool bar. After the build has started the build button is displayed as a cancel button, allowing the build to be cancelled before completion if required.

A full consistency and error check will be performed on the configuration when building the hooked ECU software. Errors, warnings and information messages will be displayed in the **Application Log** window. This can be displayed via the **View Application Log** menu command. Additionally, the output log of all build messages can be seen within the **Output** window. This can be displayed via the **View Output** menu command.

### 5.9.3 Options

The **Tools Options** menu brings up a dialog (see figure 5.42) that allows additional command line options to the EHOOKS ECU Target Support tools to be specified. This dialog is useful for quickly modifying default EHOOKS-DEV build options such as output file naming, error and warning messages and other commonly used options. Additional information on these options can be found in section 11.1 [EHOOKS-DEV Command Line Usage](#). Clicking the **Restore Default** button will cancel these additional command line options and restore to the previous default options.

---

#### NOTICE

*The build options are not saved with the EHOOKS project file, rather they are saved globally as part of the EHOOKS-DEV tool settings. They will therefore remain the same between different projects unless specifically changed within this dialog.*

---

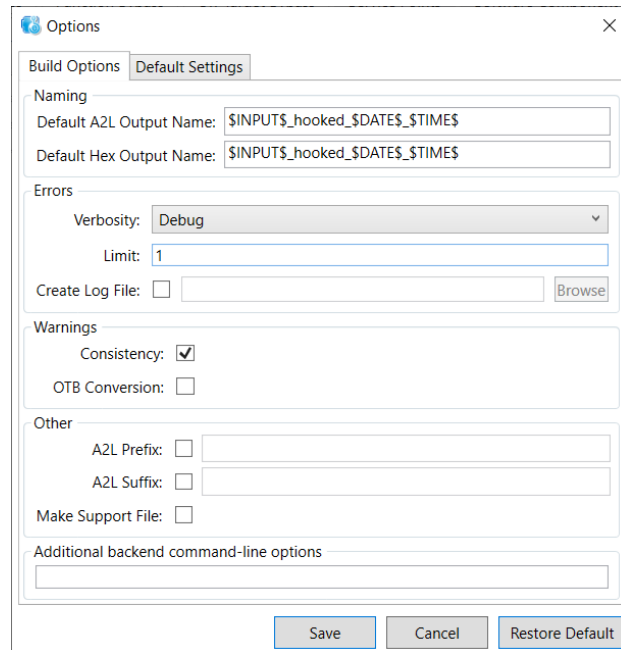


Figure 5.42: EHOOKS-DEV Options Dialog

## 5.10 Project Actions

The Project menu provides some useful options for managing an EHOOKS project. Using this menu, the following actions can be performed on the current EHOOKS project.

### 5.10.1 Convert All Paths

All paths within a project can be converted to be absolute or relative by clicking the relevant menu item in **Project Convert All Paths**, as shown in figure 5.43.

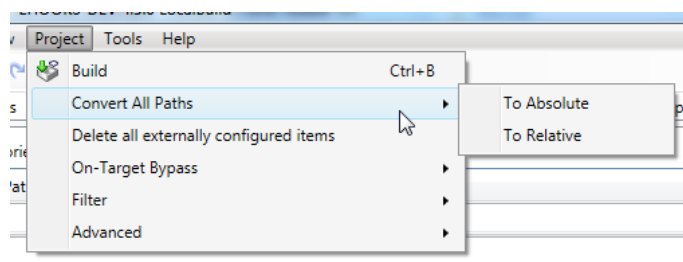


Figure 5.43: Convert All Paths Menu Item

The following paths will be modified using this action:

- A2L Input File
- A2L Output File
- ECU Image Input File
- ECU Image Output File
- All Include Directories

- All Files to Build
- All Pre- and Post-Build Scripts

#### 5.10.2 Delete all externally configured items

By clicking on **Project -> Delete all externally configured items**, the user can delete all items that have been configured externally, for example by using a tool such as ASCET or Simulink to create an On-target bypass configuration.

#### 5.10.3 On-target bypass

The use of the **Project -> On-Target bypass** options is described in detail in section [5.9.1 Consistency Checking](#).

#### 5.10.4 Filter files

Project filter files provide a way to exclude certain ECU variables and ECU processes from appearing within the EHOOKS configuration tool for hooking. This feature is designed to enable a project to implement specific rules to enforce that EHOOKS is not used to hook certain specified variables.

A project filter file can be created by selecting the **Project -> Filter -> New Filter File...** menu item. In the resulting file selection dialog, the name and location for the filter file should be specified. EHOOKS will then create a new empty project filter file using the specified file and load it into the project configuration.

A project filter file can be loaded by selecting the **Project -> Filter -> Load Filter File...** menu item. In the resulting file selection dialog, the required project filter file should be selected. EHOOKS will then load the specified project filter file into the project configuration.

A project filter file can be unloaded by selecting the **Project Filter Unload Filter File** menu item.

Once a project filter file has been loaded into the project configuration, its contents can be edited by selecting the **Project -> Filter -> Edit '<Filename>'** menu item. This will launch the filter-file editor summary dialog as shown in figure 5.44. This displays the number of ECU measurements and ECU process that have been filtered out of the EHOOKS project within the specified filter file.

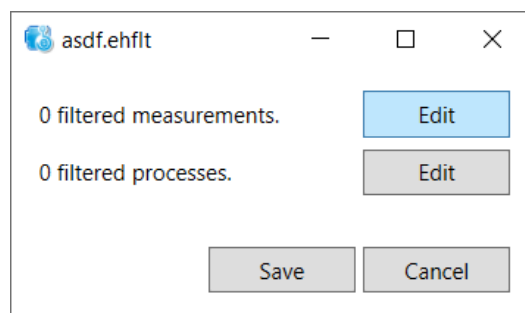


Figure 5.44: Filter File Editor Summary Dialog

To change the ECU measurements (variables) specified in a filter file click the associated **Edit** button. This launches the filter file variable selection editor as shown in figure 5.45.

The left-hand column allows the displayed variable list to be filtered by the A2L file function groups (including filtering by the input, output and local measurements to a function). To

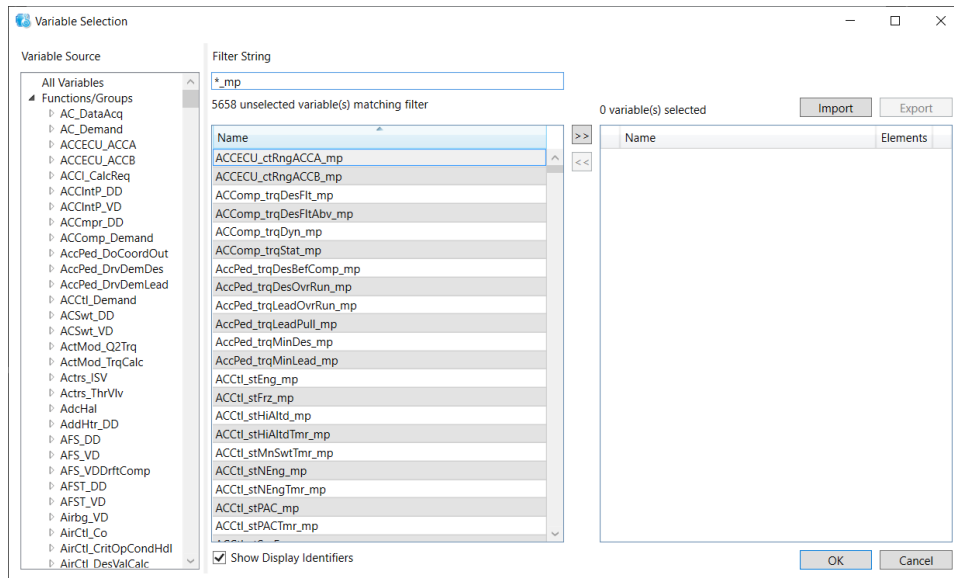


Figure 5.45: ECU Measurements Filter File Editor

quickly move to the desired function group, click on an entry in the function group list to move focus and then begin typing.

The middle-column displays a variable list which can be filtered by typing a filter string into the text box. The filter string can include the wildcard characters `?` and `*`. `?` will match any single character and `*` will match any number of characters. Note that the filter string contains an implicit `*` wildcard at the end of the string. The character `$` can be used to match the end of a variable name. As the filter string is updated, the variable list will dynamically update to show any matching ECU variables that can be added to the filter file.

The right-hand column shows the ECU variables that have been added to the filter file. ECU variables can be moved in and out of this list using the `>>` and `<<` buttons, or by pressing `Ctrl+right-arrow` and `Ctrl+left-arrow`, respectively. Additionally, double clicking on an entry in either list will move it to the other list. If an array variable has been added to the filter file, the elements of the array to be filtered can be configured by clicking on the `...` button. The number of filtered elements is then shown in the right-hand column.

The **Show Display Identifiers** check box allows the list of variable names to be changed to show the `DISPLAY_IDENTIFIER` fields from the A2L file.

To change the ECU processes specified in the filter file, click the associated **Edit** button. This launches the filter file ECU process selection editor as shown in figure 5.46.

The left-hand column contains a list of the ECU processes that can be filtered, while the right hand column contains a list of ECU processes that have been added to the filter file. ECU processes can be moved between the two columns using the buttons `>>` and `<<`. Additionally, double clicking on an entry in either list will move it to the other list.

The dialog offers a filter to help locate the desired ECU process; this behaves in the same way as for the variable selection dialog.

In some situations, the ECU software provider may have prepared some ECU processes to allow them to be bypassed, but indicated that their names should not be displayed in the EHOOKS-DEV user interface. In this case, such functions can be added to the filter file by



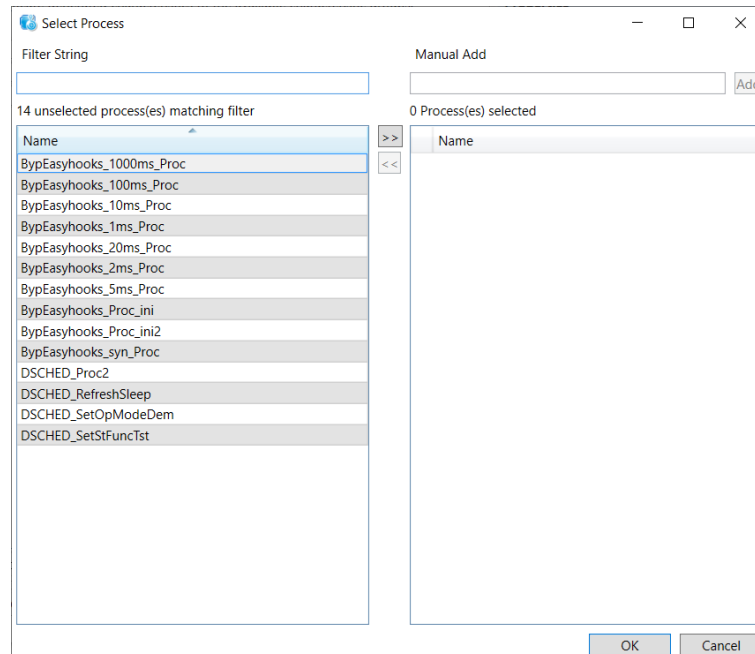


Figure 5.46: ECU Processes Filter File Editor

typing their name into the **Manual Add** text field and clicking the **Add** button.

Once edits to the filter file have been made, they can be committed by clicking the **Save** button within the filter file editor summary dialog.

---

**NOTICE**

*The loaded filter file saved with the EHOOKS project configuration, it is not saved globally as part of the EHOOKS-DEV tool settings. To configure a standard filter file to be used for all projects it is recommended to use the `--filterfile` command-line option. This can be specified as a global tool setting within the build options dialog, see [5.9.3 Options](#).*

---

#### 5.10.5 Advanced actions

The use of the **Project Advanced** options is described in detail in section [11.5 Advanced Project Options](#).

## 6 Working with Hooked ECU Software in INCA

This section provides general details on how to work with the EHOOKS-created hooked ECU software. This section also includes details on how to use EHOOKS-CAL and EHOOKS-BYP to enable the usage of the EHOOKS-created hooks within the ECU software. Depending on the EHOOKS license in use and the setting for **Licensing Mode** within the EHOOKS project settings configuration, the use of EHOOKS-CAL or EHOOKS-BYP may or may not be required.

In general, EHOOKS produces a standard ECU HEX and A2L file and therefore the usage of the hooked ECU software with INCA is little changed from the original software. The software must be flashed to the ECU in the normal manner and the experiment setup is basically unchanged.

---

### NOTICE

*As EHOOKS changes the ECU software, it is often necessary to ensure that the code-space checksums of the ECU software are disabled or at least do not trigger an ECU reset. This is sometimes done by the ECU software provider when they prepare the ECU software for EHOOKS.*

*However, these checksums are more often disabled by specific calibration data settings. In this case, please contact the ECU software provider for details of which calibration parameters to set and which values to use.*

*If applying the calibration data set from a non-EHOOKS project to an EHOOKS-created hooked ECU software project, ensure that the necessary calibration data settings to disable the checksums are set correctly again after the copy of the calibration data set.*

---

### 6.1 Run-Time Hook Control and Monitoring

If enablers have been configured, hooks or service points added to the ECU software can be controlled at run-time. Enablers can be added to the INCA experiment in the normal manner for any calibration parameter (using the variable selection). For convenience, EHOOKS creates A2L function groups to make locating the hook enablers easy and efficient (see section [6.6 A2L Function Groups](#))

The status of the enablers can be managed at run-time simply by switching their values between TRUE and FALSE as necessary in the INCA experiment environment.

The run-time behavior of the enablers is as follows:

- Variable bypass

If enabled, the bypass value will be written to the hooked ECU variable. Otherwise, the original ECU calculation is written to the hooked ECU variable.

- Function bypass

If enabled, the ECU function is bypassed (i.e. not executed). Otherwise, the ECU function is executed as normal.

- On-target bypass function

If enabled, the on-target bypass function is executed on the ECU to calculate bypass values. Otherwise, the user provided bypass function is not executed on the ECU.

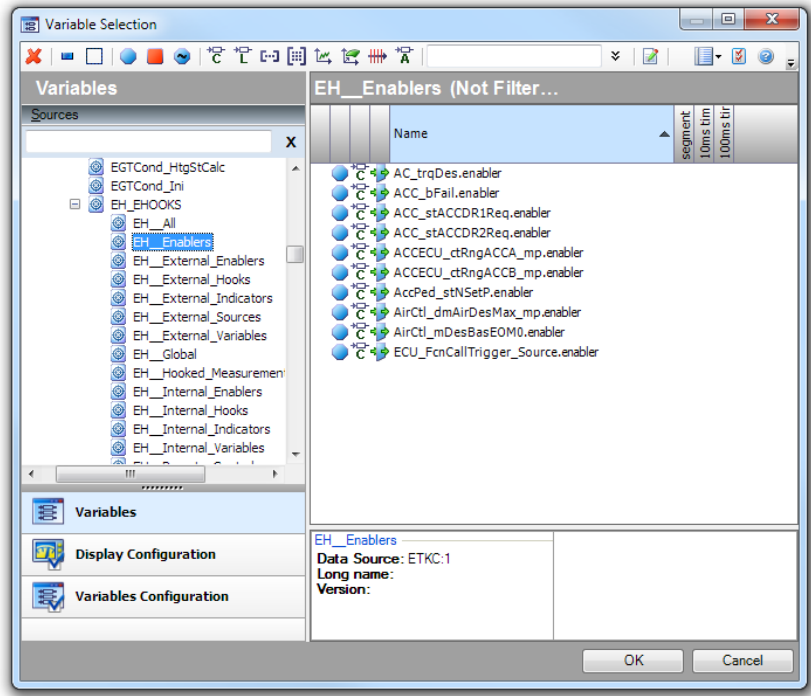


Figure 6.1: Adding Hook Enablers to the INCA Experiment

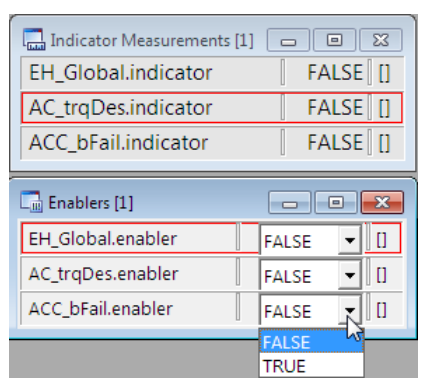


Figure 6.2: Managing Enabler Calibration Characteristics in INCA

In the case that the on-target bypass function is not executed:

- If the variable bypass enablers are set to true, the most recent valid return values from the on-target bypass function are written to the hooked ECU variables associated with the on-target bypass function.
- If the variable bypass enablers are set to true and there has not been a valid return value from the on-target bypass function, the original ECU calculations are written to the hooked ECU variables associated with the on-target bypass function
- Service Points

If a service point is enabled, the selected ECU process is bypassed and the service point is executed instead. If the service point is disabled, the ECU process is executed as normal.

The logic to control whether a specific hook/service point is enabled or disabled is as follows:

- If a global enabler is configured and is set to false, all hook/service point are **disabled**.
- If a global enabler is configured and is set to true, or a global enabler is not configured:
  - If any group enabler associated with the hook/service point is true, or the hook/service point specific enabler is true, or the hook/service point specific control variable (see section [3.2.2.3 Control Variables](#)) is true, the hook is **enabled**.
  - If all group enablers associated with the hook/ service point are false, the hook/ service point specific enabler is false, and the hook/ service point specific control variable is false, the hook/ service point is **disabled**.

If configured, the status of the enablers can be monitored using the indicator measurements created by EHOOKS. For convenience, EHOOKS creates A2L function groups to make locating the indicator measurements easy and efficient (see section [6.6 A2L Function Groups](#)).

---

**NOTICE**

*If an enabler is changed but the associated indicator does not mirror the change, the most likely reason is that the ECU software is not currently executing the hook code inserted by EHOOKS. When this situation is observed, it is possible to use the advanced write location functionality within the associated hook to force the hook code to be executed (see sections [3.2.2.6 Forced-Write Mechanisms](#), [Configuring Properties of a Variable Hook](#), and [\[Properties of an On-Target Bypass Function\]](#)).*

---

When indicators are configured, EHOOKS automatically creates the following additional diagnostic counters for use at run-time.

- <indicator>.hookDiag

This counter is incremented every time the EHOOKS-inserted hook code runs.

- <indicator>.matchDiag

This counter is incremented every time the EHOOKS-inserted hook code runs and the address passed by the hook matches that of the hooked measurement variable. I.e. it is incremented when a write to a hooked variable is correctly detected.

- <indicator>.bypassDiag

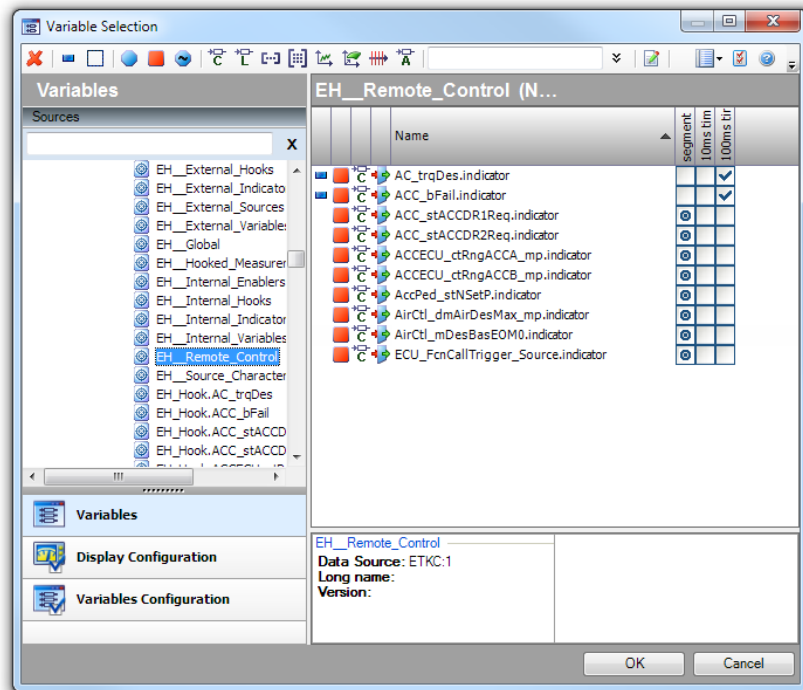


Figure 6.3: Adding Indicators to the INCA Experiment

This counter is incremented every time the bypass value is written into the hooked ECU measurement variable.

## 6.2 Offset Hooks

When a hook has been configured as an offset hook, EHOOKS creates an additional calibration characteristic to allow run-time control over whether the bypass value is added or subtracted to the ECU calculated value. Offset control characteristic names are based on the hooked ECU variable name prefixed with `EH_ctrl_`.

For convenience, EHOOKS creates an A2L function group, called `EH_Offset_Controls`, containing all the offset control calibration characteristics (see section 6.6 A2L Function Groups).

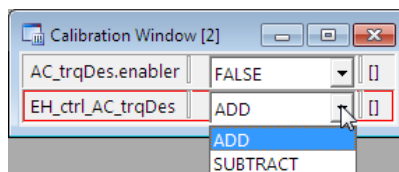


Figure 6.4: Managing an Offset Hook in INCA

### NOTICE

For a constant bypass hook, an offset control characteristic is not created as the constant is a signed value and therefore the constant value is always added to the ECU calculation. Subtraction is achieved by using a negative constant value.

### 6.3 Backup Measurement Copies

When a hook has been configured to include a backup measurement copy, EHOOKS will create a new measurement variable. EHOOKS will ensure that this measurement variable is kept up to date with the current value of the original ECU calculation for the hooked variable. This allows the comparison of the original ECU calculation and the bypass value. The backup measurement copy names are based on the hooked ECU variable name prefixed with EH\_copy\_.

For convenience, EHOOKS creates an A2L function group, called EH\_\_Copies, containing all the backup measurement copies (see section 6.6 A2L Function Groups).

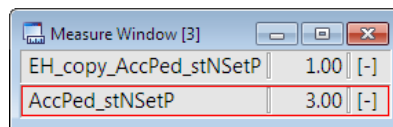


Figure 6.5: Monitoring Backup Measurement Copies in INCA

### 6.4 Safety Checks

When a hook has been configured to include safety checks, EHOOKS adds additional code to monitor the hook for run-time errors. If a run-time error is detected by this code, it will be indicated via a safety detection failure measurement. One safety detection failure measurement is created for each hooked ECU variable configured to include safety checks. Its name will be based on the hooked ECU variable name prefixed with EH\_err. An error is indicated if this measurement has the value of 0x12.

When EHOOKS detects a run-time error in a hook, it will be automatically disabled so that the associated hooked ECU variable is no longer bypassed. This can be overridden using the error override calibration characteristic that EHOOKS creates. An error override characteristic will be created for each hooked ECU variable configured to include safety checks. Its name will be based on the hooked ECU variable prefixed with EH\_eovd\_. Setting an error override characteristic to TRUE means that the associated hook will continue to be enabled even if errors are detected by the safety checks. Setting an error override characteristic to FALSE means that the associated hook will be automatically disabled if errors are detected by the safety checks.

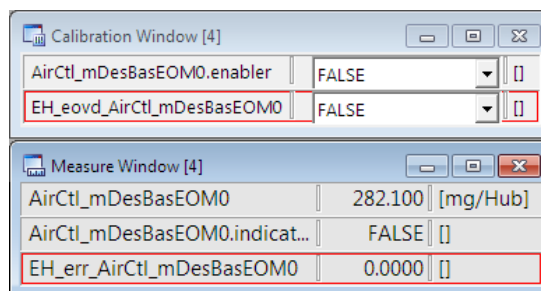


Figure 6.6: Monitoring and Managing Safety Checks in INCA

### 6.5 Using EHOOKS-CAL and EHOOKS-BYP to Work with Hooks in INCA

To work with hooked ECU software created by EHOOKS using the locked licensing mode the EHOOKS Hook Un-locker tool (EHOOKS-CAL/EHOOKS-BYP) must be used. The original functionality of the ECU software is unaffected by this requirement. Without using

EHOOKS-CAL or EHOOKS-BYP, the modified features of the hooked ECU software added by EHOOKS will be disabled and cannot be used.

---

*NOTICE*

*After the ECU is reset, the modified features of the hooked software added by EHOOKS will function for a brief period (typically the first 3 - 5 minutes). This provides time to connect EHOOKS-CAL/EHOOKS-BYP to the ECU (via INCA) to enable the functionality to be continuously unlocked. This also allows experiments that need to bypass hooked ECU variables during ECU start-up, initialization and cranking to be easily performed.*

---

The EHOOKS-CAL and EHOOKS-BYP are implemented via the single EHOOKS Hook Un-locker product. The operation mode depends on the license key installed. EHOOKS-CAL and EHOOKS-BYP allow different types of hooks within the hooked ECU software to be unlocked as follows:

- **EHOOKS-CAL:** Enables the use of constant, calibration and NOP hooks within the EHOOKS-created hooked ECU software, but will not unlock any external or on-target bypass hooks nor any service points.
- **EHOOKS-BYP:** Enables the use of all hook types within the EHOOKS-created hooked ECU software.

The EHOOKS Hook Un-locker is implemented as an INCA add-on and must therefore be used in conjunction with a running INCA experiment.

How to work with EHOOKS-CAL/EHOOKS-BYP

- Create hooked ECU software using EHOOKS with the licensing mode set to **Locked Hooks**.
- Import the EHOOKS-created hooked ECU software (both A2L and HEX file) into INCA.
- Flash the ECU with the hooked ECU software in the normal manner, typically using INCA/ProF.
- Set up an INCA experiment and connect to the ECU.
- Begin measurement and switch to the working page.
- Launch the EHOOKS hook un-locker.
- Click **Start Unlocking**.
- The hooks will be unlocked and it is then possible to continue working with INCA in the normal way using the modified features of the hooked ECU software.

The EHOOKS Hook Unlocker interface consists of two major elements. The **Status** section gives feedback to indicate whether the hooks have been unlocked. The **Log** section gives feedback about each step of the interaction between the EHOOKS Hook Unlocker and INCA.

In addition, the EHOOKS Hook Unlocker will place an icon in the system tray to indicate the current status. Different icons indicate each different status as shown in figure 6.8.

The **Start Unlocking** button will initiate the hook unlocking process. Prior to clicking this, INCA should be loaded with an experiment with measurement running and the working page

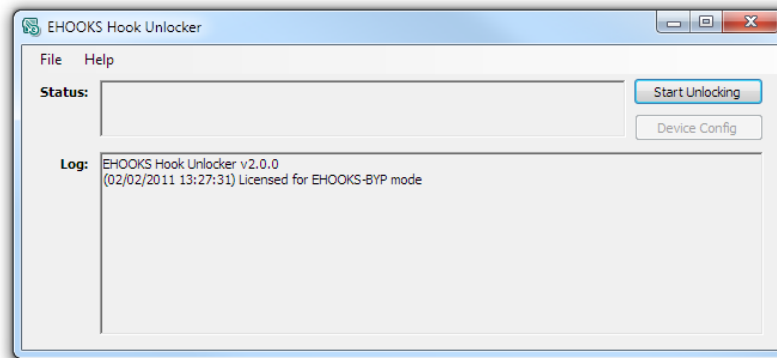


Figure 6.7: EHOOKS Hook Unlocker (EHOOKS-CAL/EHOOKS-BYP)

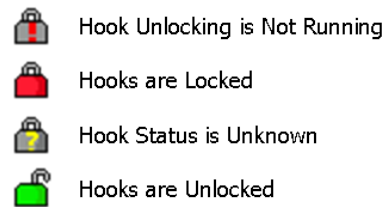


Figure 6.8: EHOOKS Hook Unlocker System Tray Icons

selected. The Log window will give feedback if any of these conditions are not met.

The **Start Unlocking** button will change to a **Cancel** button once clicked. The **Cancel** button allows the EHOOKS Hook Unlocker to be cleanly disconnected from INCA.

---

**NOTICE**

*After the EHOOKS Hook Unlocker has been used to enable hooks in the ECU, it is possible to disconnect the EHOOKS Hook Unlocker (via the **Cancel** button). The hooks will remain enabled until the ECU is rebooted, after which the EHOOKS Hook Unlocker must be used again to re-enable the hooks.*

*This can be important as the EHOOKS Hook Unlocker uses the INCA COM API which allows one client to be connected at a time. By disconnecting the EHOOKS Hook Unlocker it is therefore possible to connect other tools to the INCA COM API to work with the EHOOKS-created hooks.*

---

The **Device Config** button allows the selection of the INCA raster to be used for acquisition of measurement data by the EHOOKS Hook Unlocker. If no feedback is received after clicking the **Start Unlocking** button, it may be necessary to change the ECU raster used for acquisition of measurement data.

To change the device configuration measurement should first be stopped in INCA. Once the device configuration is set up, measurement can be started again within INCA.



**NOTICE**

*It is important to ensure that the A2L file loaded into the INCA database is the one created by EHOOKS at the same time the hooked HEX file was created. If this is not the case then it is likely that the EHOOKS-created hooks in the hooked ECU software will not be successfully unlocked.*

*To ensure the correct A2L file is loaded, select the A2L entry in the INCA database and then **right-click Update** and then select the correct A2L file to be loaded into the INCA database.*

For hooked ECU software created with the **Open Hooks** licensing mode of EHOOKS, it is not necessary to use the EHOOKS Hook Unlocker.

## 6.6 A2L Function Groups

The EHOOKS-created hooked ECU software A2L file contains a number of new function groups to make it easy and efficient to locate the new EHOOKS measurements and calibration parameters, so they can be quickly added into the INCA experiment.

Table 1: EHOOKS Hooked Software A2L Function Groups

Group Name	Group Contents
EH_EHOOKS	
EH__All	Contains all of the measurements and calibration characteristics created by EHOOKS-DEV
EH__Copies	Contains all backup measurement copies created for hooked ECU variables
EH__Enablers	Contains all hook enabler calibration characteristics
EH__Error_Flags	Contains all safety detection failure measurement for hooked ECU variable configured with safety checks
EH__Error_Overrides	Contains all error override calibration characteristics for hooked ECU variable configured with safety checks
EH__External_Enablers	Contains all enabler calibration characteristics for hooked ECU variables configured for external bypass
EH__External_Hooks	Contains all calibration characteristics and measurements related to hooked ECU variables configured for external bypass – includes enablers, indicators, offset control, safety checks and the hooked ECU variable itself
EH__External_Indicators	Contains all indicator measurements for hooked ECU variables configured for external bypass

Group Name	Group Contents
EH__External_Variables	Contains the hooked ECU variables configured for external bypass
EH__Global	Contains the global enabler and indicator, if configured, and a hook count calibration characteristic. Note: The hook count calibration characteristic can be used to indicate how many ECU variable hooks are placed into the ECU software. Changing the value of this calibration characteristic has no effect.
EH__Group_Enablers	Contains enabler characteristics created for hook/service point/OTB function groups
EH_Group.<group>	Contains the enabler characteristic created for the group called <group> and the per hook/service point/OTB function enabler characteristics for the things in the group called <group>
EH__Hooked_Measurements	Contains all hooked ECU variables
EH__Internal_Enablers	Contains all enabler calibration characteristics for hooked ECU variables configured for internal bypass (i.e. constant, calibration or on-target bypass)
EH__Internal_Hooks	Contains all calibration characteristics and measurements related to hooked ECU variables configured for internal bypass (i.e. constant, calibration or on-target bypass) – includes enablers, indicators, offset control, safety checks and the hooked ECU variable itself
EH__Internal_Indicators	Contains all indicator measurements for hooked ECU variables configured for internal bypass (i.e. constant, calibration or on-target bypass)
EH__Internal_Variables	Contains the hooked ECU variables configured for internal bypass (i.e. constant, calibration or on-target bypass)
EH__Offset_Controls	Contains the offset control calibration characteristics for hooked ECU variables hooked for offset bypass
EH__OTB_Function_Outputs	Contains the on-target bypass output variable buffer measurements for each ECU variable configured for on-target bypass.
EH__Remote_Control	Contains all indicator measurements

Group Name	Group Contents
EH__Service_Points	Contains the enabler characteristics and indicator measurements configured for service points.
EH__Source_Characteristics	Contains the new calibration characteristics to be used as the bypass value for hooked ECU variables configured for calibration bypass
EH_Hook.<ECU Variable>	For each hooked ECU variable, contains all measurements and calibration characteristics related to the hook – includes enablers, indicators, offset control, safety checks and the hooked ECU variable itself
EH_User_Func.<OTB-Function>	For each on-target bypass function (whether coming from Simulink, ASCET or manually integrated on-target bypass functions), contains all measurements and calibration characteristics related to the function - includes enablers, indicators and the associated hooked ECU variables
EH_EHOOKS_OnTargetBypass	This group contains measurements and calculation parameters created from any User Definitions Files and by the EHOOKS-DEV Simulink Integration Package.
EH_OTH_BypassModelControl	Contains all enablers calibration characteristics and indicator measurements configured for Simulink based on-target bypass hooks
EH_OTB_FunctionEnablers	Contains the enablers calibration characteristics for the Simulink configured on-target bypass functions
EH_OTB_FunctionIndicators	Contains the indicator measurements for the Simulink configured on-target bypass functions
EH_VariableEnablers	Contains the enablers calibration characteristics for the variables hooked for bypass by Simulink configured on-target bypass functions
EH_VariableIndicators	Contains the indicator measurements for the variables hooked for bypass by Simulink configured on-target bypass functions
EH_OTB_Characteristics	Contains all new calibration characteristics introduced within on-target bypass functions

Group Name	Group Contents
EH_OTB_ScalarCharacteristics	Contains all new scalar calibration characteristics introduced within the Simulink models used for on-target bypass
EH_OTB_ComplexCharacteristics	Contains all new axis, map, curve and value block calibration characteristics introduced within the Simulink models used for on-target bypass.
EH_OTB_AxisCharacteristics	Contains all new axis calibration characteristics introduced within the Simulink models used for on-target bypass
EH_OTB_CurveCharacteristics	Contains all new curve calibration characteristics introduced within the Simulink models used for on-target bypass
EH_OTB_MapCharacteristics	Contains all new map calibration characteristics introduced within the Simulink models used for on-target bypass
EH_OTB_ValueBlockCharacteristics	Contains all new value block calibration characteristics introduced within the Simulink models used for on-target bypass
EH_OTB_Measurements	Contains all new measurements introduced within on-target bypass functions
EH_OTB_ScalarMeasurements	Contains all new scalar measurements introduced within the Simulink models used for on-target bypass
EH_OTB_ArrayMeasurements	Contains all new array measurements introduced within the Simulink models used for on-target bypass

## 7 Creating and Working with Simple Internal Bypass

There are two kinds of simple internal bypass – constant and calibration.

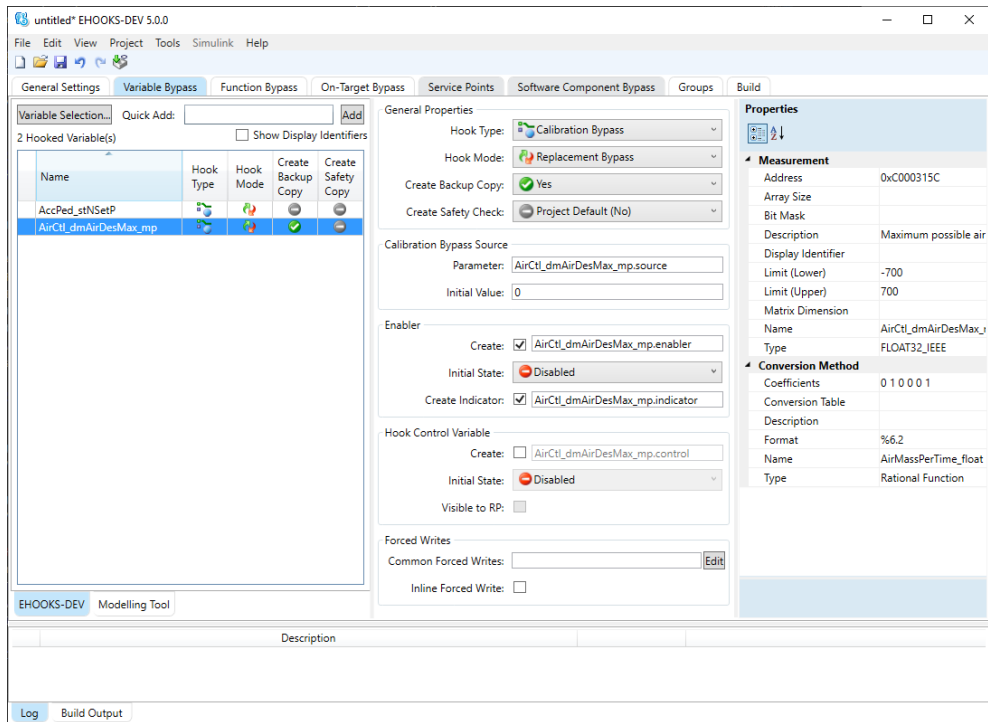


Figure 7.1: Simple Internal Bypass Configuration

A constant value bypass uses a fixed, build-time, constant as the bypass value for a hooked ECU variable. In this case, when the set of configured hook enablers evaluate to true (and assuming no safety check failure) the configured constant value is used as the bypass value for the hooked ECU variable. Figure 7.2 shows a constant value bypass running in the INCA experiment environment where the variable AccPed\_stNSetP is being hooked by EHOOKS with a constant value bypass of 12.

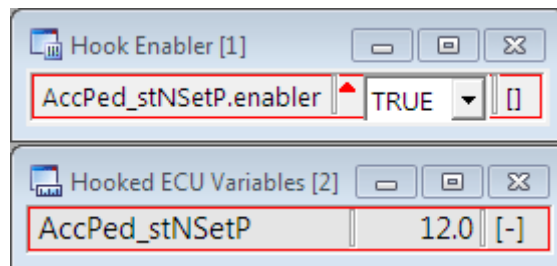


Figure 7.2: Constant Value Bypass

A calibration characteristic bypass allows a calibration characteristic created by EHOOKS-DEV to be used to provide the bypass value for a hooked ECU variable. In this case, when the set of configured hook enablers evaluate to true (and assuming no safety check failure), the current value of the configured calibration characteristic is used as the bypass value for the hooked ECU variable. Figure 7.3 shows a calibration bypass running in the INCA experiment environment, where the variable AirCtl\_dmAirDesMax\_mp is being hooked by EHOOKS and bypassed with the new calibration parameter AirCtl\_dmAirDesMax\_mp.source being used as an offset bypass (rather than a replacement bypass).

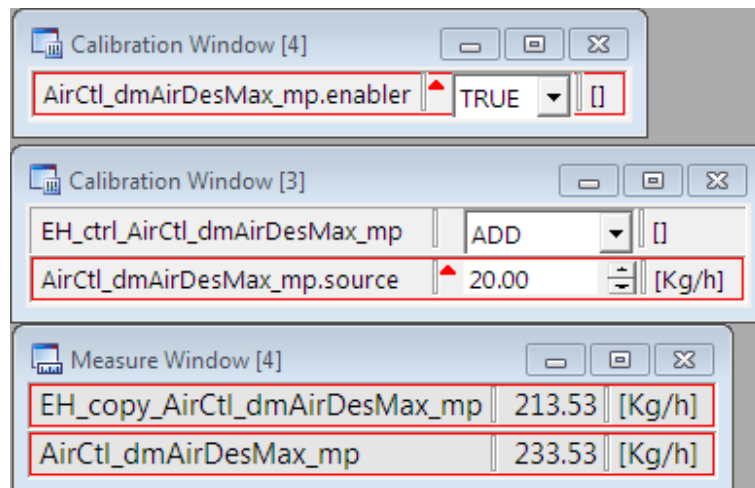


Figure 7.3: Calibration Bypass

## 8 Creating and Working with External Bypass

### 8.1 Hook based bypass (HBB)

Configuring a variable for external hook based bypass causes EHOOKS-DEV to add the necessary ASAP1B annotation to the associated entry in the A2L file. The steps to work with external bypass hooks with EHOOKS are as follows (for full details on the usage of INTECRIO please refer to the INTECRIO user documentation – EHOOKS does not change the usage of INTECRIO at all):

1. Configure the EHOOKS-DEV project to hook the desired ECU variables for external bypass (it is, of course, possible to hook other variables for other types of bypass – constant, calibration or on-target – within the same project configuration).
2. Use EHOOKS-DEV to build the new ECU software.
3. Load the EHOOKS-DEV-generated A2L file for the ECU software into the INTECRIO hardware item to allow the newly hooked variables to be selected from the signal selection configuration.
4. Configure the bypass experiment in the usual way and build the bypass software with INTECRIO.
5. Load the EHOOKS-DEV-generated A2L/HEX file into INCA.
6. Re-flash the ECU with the EHOOKS-DEV-generated HEX file.
7. Perform the bypass experiment in the usual way, either using INCA-EIP or INCA and the INTECRIO experiment environment. The EHOOKS-DEV-created bypass hooks must be unlocked using the EHOOKS Hook Unlocker.

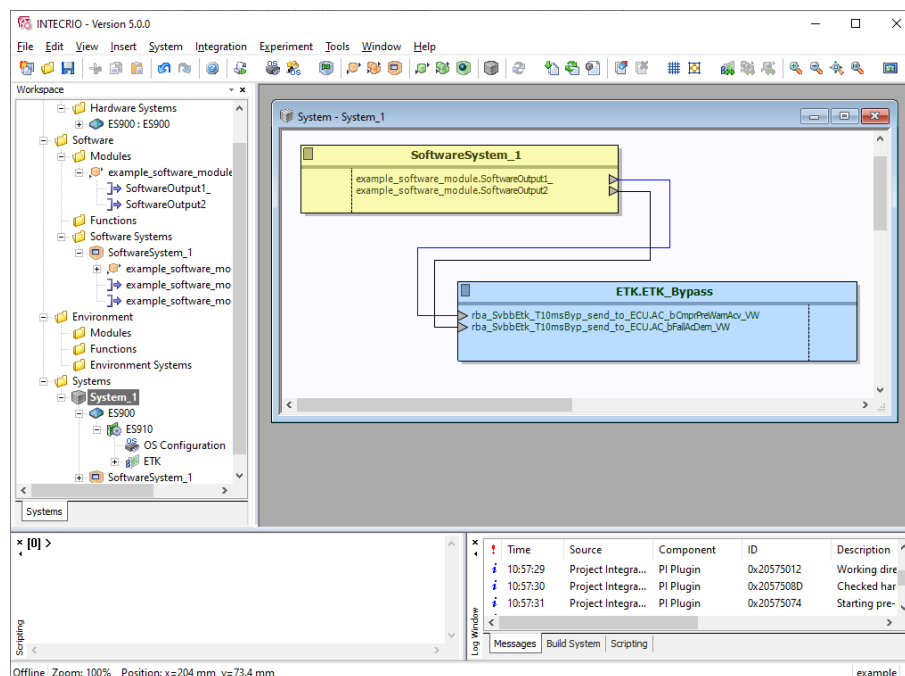


Figure 8.1: External Bypass

External bypass hooks perform the same as other types of hooks: when the set of configured hook enablers evaluates to true (and assuming no safety check failure), the current value

being calculated on the external rapid prototyping hardware is used as the bypass value for the hooked ECU variable.

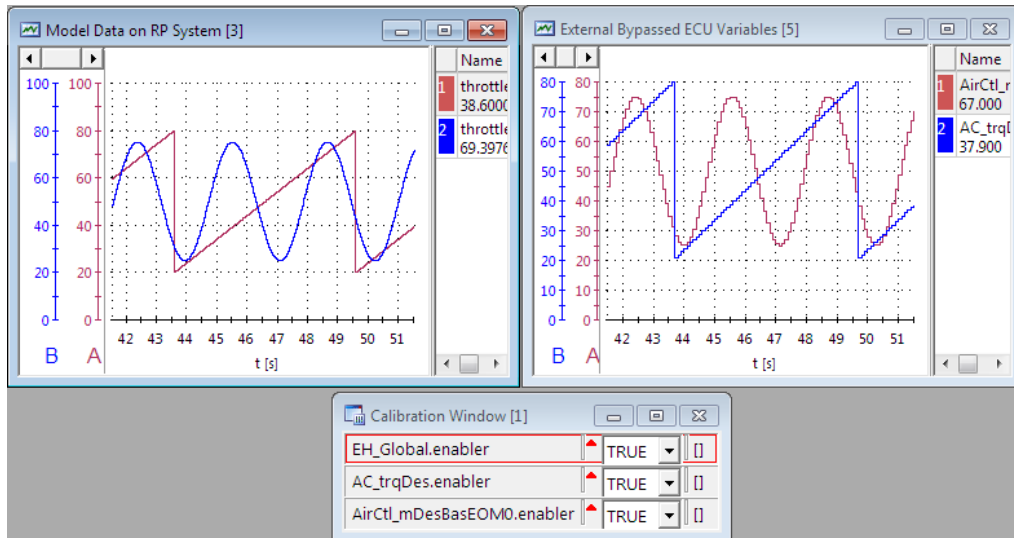


Figure 8.2: External Bypass Experiment in INCA

## 8.2 Service based bypass (SBB)

Configuring a process or function group for service based bypass (bypassing the original ECU process or function group with a services point), causes EHOOKS-DEV to add the necessary service point information to the A2L file. The steps to work with external service based bypass with EHOOKS are as follows:

1. Configure the EHOOKS-DEV project to insert the desired service points into the ECU software as described in section [5.6 Service Points Tab](#)
2. Follow steps 2 to 7 described in section [8.1 Hook based bypass \(HBB\)](#)



## 9 Creating and Working with On-Target Bypass

### 9.1 Introduction

Configuring on-target bypass enables new bypass algorithms to be introduced directly into the ECU software. EHOOKS provides a standardized interface to make it easy to integrate bypass algorithms directly into the ECU software, along with measurement and calibration capabilities for the new bypass algorithm.

How to configure on-target bypass

- Step 1: Configure On-Target Bypass Hooks

Configure the EHOOKS-DEV project to hook the desired ECU variables for on-target bypass [Optional].

- Step 2: Configure On-Target Bypass Functions

Configure the EHOOKS-DEV project to add the desired on-target bypass functions and the associated input and output ECU variables. [Any output variables configured will be automatically configured by EHOOKS-DEV as on-target bypass hooks if they are not already configured with the above step].

- Step 3: Develop the On-Target Bypass Software

Develop the C code (and any associated C header, library and object files) to implement the configured on-target bypass functions. Optionally, create one or more user definition files to define new measurements and calibration characteristics needed to measure and calibrate the on-target bypass functions.

- Step 4: Add Build Files to the EHOOKS-DEV project configuration

Add the associated on-target bypass files to the EHOOKS-DEV project configuration.

- Step 5: Build and Run the EHOOKS-Created Software

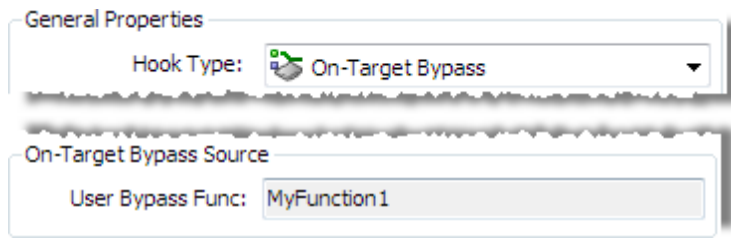
Use EHOOKS-DEV to build the new ECU software, load the EHOOKS-DEV-generated A2L/HEX file into INCA, re-flash the ECU with the EHOOKS-DEV generated HEX file and finally perform the bypass experiment in the usual way using INCA.

The key steps will now be described in more detail. To illustrate the process, a consistent example will be used throughout this section. This example involves using an on-target bypass called MyOTBFunc to calculate the bypass value for AC\_trqDes and InjCrv\_qPii1Des[0] using as inputs the ECU variables AccPed\_stNSetP and AirCtL\_mDesBasEOM0.

### 9.2 Step 1: Configure On-Target Bypass Hooks

The variables to be hooked for on-target bypass should be added to the variable bypass configuration tab within EHOOKS-DEV, with a hook type of **On-Target Bypass**. This can be achieved in one of two ways.

First, the variable can be added to the variable bypass configuration tab as described in section [5.2.1 Selecting Variables to Be Hooked](#). Then the hook-type should be set to **On-Target Bypass**. Any on-target bypass functions that include the variable as an output will be displayed in the read-only On-Target Bypass Source field.

**NOTICE**

*EHOOKS-DEV allows one on-target bypass function to be used to calculate the bypass values for an arbitrary number of ECU variables. It is not necessary to have a separate on-target bypass function for each ECU variable.*

Second, any variables configured using Step 2 below will automatically be added to the variable bypass configuration tab with the correct properties, unless they are already hooked.

### 9.3 Step 2: Configure On-Target Bypass Functions

Each on-target bypass function must be added to the on-target bypass tab and the associated inputs, outputs and properties configured as described in section [5.4 On-Target Bypass Tab](#).

**NOTICE**

*If the on-target bypass is being configured using ASCET or Simulink, the on-target bypass function should not be added to the on-target bypass tab manually. ASCET or Simulink should be used to directly add the corresponding hooks and configuration (see section [10 Creating and Working with On-Target Bypass from Simulink](#) or the ASCET-SE EHOOKS Add-on User Guide for full details – this can be found in the ASCET installation directory within the folder <ASCET\_INSTALL\_DIRECTORY>\target\trg\_ehooks\documents). These on-target bypass functions will appear within the On-Target Bypass tab under the **Modeling Tool** list available at the bottom of the on-target bypass list.*

### 9.4 Step 3: Develop the On-Target Bypass Software

EHOOKS-DEV creates a simple C interface for each on-target bypass function, to make development and integration straightforward. This interface abstracts the details of when/how the on-target bypass function is executed allowing EHOOKS to take complete responsibility for the details of providing input parameters to the function and using the output parameters for bypass values.

EHOOKS-DEV will display an example code template that can be used as a reference for the creation of the on-target bypass implementations. The following provides details regarding this code structure.

#### 9.4.1 On-Target Bypass Function Input and Output Parameters

The on-target bypass interface data structures are generated by EHOOKS-DEV in a file called "UserBypassFuncs.h". This file should therefore be included in any on-target bypass function code.

The signature of an on-target bypass function in EHOOKS is of the form:

```
int My0TBFunc(int EH_context)
```

However, it is more convenient to use the macro:

```
EH_USER_BYPASS_FUNC()
```

If this macro is used the declaration of the on-target bypass function becomes:

```
EH_USER_BYPASS_FUNC(MyOTBFunc)
```

To read an input, an on-target bypass function should use:

```
EH_ARG_GET_<input-var-name>(EH_context)
```

Where <input-var-name> is the name of an input variable read by the on-target bypass function.

To write an output, the on-target bypass function should use:

```
EH_ARG_PUT_<output-var-name>(<value>)
```

Where <output-var-name> is the name of an output variable written by the on-target bypass function and is the value to be written.

---

**NOTICE**

*The EHOOKS on-target bypass interface described in sections 9.4.1 and 9.4.2 was updated in EHOOKS V3.0. Any on-target bypass functions created for use with EHOOKS-DEV V2 should therefore be modified accordingly for use with EHOOKS-DEV V3.0 onwards.*

---

#### 9.4.2 On-Target Bypass Function Implementation

The on-target bypass function simply has to read from the input parameters, perform its calculation and write the results to the output parameters. EHOOKS will take care of all other details (such as initializing the input parameters and using the output values to bypass ECU variables).

The final step of the on-target bypass function is to return a value to indicate whether or not EHOOKS should use the calculated output values for bypass. If the on-target bypass function returns a non-zero value, EHOOKS will use the output values to bypass the associated ECU variables. If the on-target bypass function returns zero, EHOOKS will not use the output values to bypass the associated ECU variable.

Source Code 1 shows a simple implementation of an on-target bypass function for the example configuration.

Source Code 1: Example Simple On-Target Bypass Function Implementation

```
/* Include EHOOKS-DEV generated header file */
#include "UserBypassFuncs.h"

EH_USER_BYPASS_FUNC(MyOTBFunc)
{
    EH_ARG_PUT_AC_trqDes( EH_ARG_GET_AccPed_stNSetP(EH_context) +
                        EH_ARG_GET_AirCtl_mDesBasEOM0(EH_context));

    EH_ARG_PUT_InjCrv_qPiI1Des__0__( EH_ARG_GET_AirCtl_mDesBasEOM0(EH_context) -
```

```

EH_ARG_GET_AccPed_stNSetP(EH_context));

/* Bypass values valid for use */
return 1;
}

```

This source code illustrates the following key points:

- Line 2: Include UserBypassFuncs.h

The header file UserBypassFuncs.h provides access to the EHOOKS data types including the data type for the on-target bypass input / output data structure and therefore should be included in all files that implement on-target bypass functions.

- Line 4: On-Target Bypass Function Prototype

EHOOKS provides a macro in the header file, UserBypassFuncs.h, to ensure the correct function prototype implementation for on-target bypass functions. This macro, EH\_USER\_BYPASS\_FUNC, takes the name of an on-target bypass function as its only parameter and this name must match a name configured within the EHOOKS-DEV project configuration - for our example MyOTBFunc. The exact form of this macro is EHOOKS port-dependent but it will expand line 4 to something similar to:

```
int MyOTBFunc(int EH_context)
```

- Lines 6 to 12: Access to on-target bypass function input and output variables  
The input and output arguments are used to read values from variables AccPed\_stNSetP, AirCtl\_mDesBasEOM0 and write values to variables AC\_trqDes and InjCrv\_qPi1Des\_\_0\_\_

---

**NOTICE**

A2L variables can contain characters that are not legal characters for C variables – in such cases the illegal characters are replaced by a double underscore by EHOOKS-DEV. E.g. InjCrv\_qPi1Des[0] would become InjCrv\_qPi1Des\_\_0\_\_.

---

- Line 13: Return the Status of the On-Target Bypass Function

A return value of 1 (or in fact any non-zero value) indicates to EHOOKS that the on-target bypass function has executed successfully and the calculated output values should be used for bypass. A return value of zero would indicate that the calculated output values should not be used for bypass.

### 9.4.3 On-Target Bypass Data Type Conversion

The input and output parameters of an on-target bypass function are in ECU implementation form (not in physical form). Any necessary type conversions between different A2L data types / quantization formulae must be performed by the on-target bypass function itself. EHOOKS provides functions to do this as follows, see also the header file ConversionFuncs.h.

- EH\_<type>\_PHYS\_TO\_IMPL\_<name>

Converts from physical form to ECU implementation form, where the physical form is in a variable of type <type>. <type> can be double, float or single.

- EH\_IMPL\_TO\_<type>\_PHYS\_<name>

Converts from ECU implementation form to physical form, where the returned physical

form is of type <type>. Where <type> can be double, float or single.

- EH\_<type>\_T0\_IMPL\_<name>

Convert a variable of type <type> to the correct ECU data type for <name>. Where <type> can be double, float, single, uint64, int64, uint32, int32, uint16, int16, uint8, int8

- EH\_IMPL\_T0\_<type>\_<name>

Converts from the ECU data type for <name> to a variable of type <type>. Where <type> can be double, float, single, uint64, int64, uint32, int32, uint16, int16, uint8, int8

The following source code example demonstrates how to use the type conversion functions to convert the inputs to physical form for use in calculation, and how to use the type conversion functions to convert the physical output values into ECU implementation type for use as bypass values.

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

EH_USER_BYPASS_FUNC(MyOTBFunc)
{
    /* Convert the input arguments into physical form. */
    float local_AccPed_stNSetP =
    EH_IMPL_T0_float_PHYS_AccPed_stNSetP(EH_ARG_GET_AccPed_stNSetP(EH_context));

    float local_AirCtl_mDesBasEOM0 =
    EH_IMPL_T0_float_PHYS_AirCtl_mDesBasEOM0(EH_ARG_GET_AirCtl_mDesBasEOM0(EH_context));

    /* Write the output values in internal form. */
    EH_ARG_PUT_AC_trqDes( EH_float_PHYS_TO_IMPL_AC_trqDes(local_AccPed_stNSetP +
                                                         local_AirCtl_mDesBasEOM0));

    EH_ARG_PUT_InjCrv_qPiI1Des_0_ ( EH_float_PHYS_TO_IMPL_InjCrv_qPiI1Des_0_ (
                                                         local_AirCtl_mDesBasEOM0 -
                                                         local_AccPed_stNSetP));

    /* Bypass values valid for user */
    return 1;
}
```

#### 9.4.4 Calling ECU functions from On-Target Bypass code

It is possible to call functions that exist in the original ECU software from on-target bypass code. If the ECU software supplier has defined ECU functions for use (See section 4 [EHOOKS-PREP Dependencies](#)), those functions can be called from on-target bypass code by including EH\_CallableFunctionMacros.h in the source file. The name and prototype of the function must be provided by the ECU software supplier.

If the ECU software supplier has defined the function "Func1", and it has a single floating point argument, it may be called by inserting the following code into an on-target bypass function:

```
/* header included for use of existing ECU functions */

#include " EH_CallableFunctionMacros.h"
```

```
#include "UserBypassFuncs.h"

EH_USER_BYPASS_FUNC(My0TBFunc)
{
    /* Convert the input arguments into physical form. */
    float local_AccPed_stNSetP = EH_IMPL_TO_float_PHYS_AccPed_stNSetP(
        EH_ARG_GET_AccPed_stNSetP(EH_context));

    /* call existing ecu function */
    EH_Func1(local_AccPed_stNSetP);

    return 1;
}
```

---

**NOTICE**

*Functions will be prefixed with EH\_ in order to avoid name clashes with other mechanisms available to the ECU software supplier for function inclusion.*

---

## 9.5 Step 4: Add the On-Target Bypass Files to Configuration

To allow EHOOKS-DEV to compile, link and merge the on-target bypass code with the ECU HEX file, additional build information must be provided. The files related to the on-target bypass implementation must be added to the EHOOKS-DEV configuration using the build tab, as explained in section [5.8.1 Configuring Build Source Files](#).

EHOOKS allows on-target bypass functions to be measured and calibrated in the same way as the normal ECU software. However, EHOOKS must be informed about the measurements and calibration characteristics that are needed. User definition files can be added to the EHOOKS configuration to enable the creation of the measurements and calibration characteristics. EHOOKS supports the creation of scalar measurements and scalar, array, map and curve calibration characteristics.

Once defined in a user definition file, EHOOKS will create the necessary code and A2L file entries to implement the required measurements and calibration characteristics. EHOOKS will include implementation details about the calibration characteristics in the header file `Characteristics.h`; this file must be included by any on-target bypass files that want to access the created calibration characteristics.

### 9.5.1 Creating a User Definition File

A user definition file can be created either by hand (see section [EHOOKS-DEV User Definition Files](#)) or using the built-in user definition file editor within the Build Tab.

---

**NOTICE**

*If the on-target bypass is being configured using ASCET or Simulink then it is not necessary to manually create a user definition file. Instead, ASCET or Simulink should be used to directly specify the new measurements and calibration parameters (see section [10 Creating and Working with On-Target Bypass from Simulink](#), or the [ASCET-SE EHOOKS Add-on User Guide](#) for full details – this can be found in the ASCET installation directory within the folder <ASCET\_INSTALL\_DIRECTORY>\target\trg\_ehooks\documents).*

---

To launch the user definition editor the Edit button should be clicked next to the associated

user definition file as shown in figure 9.1

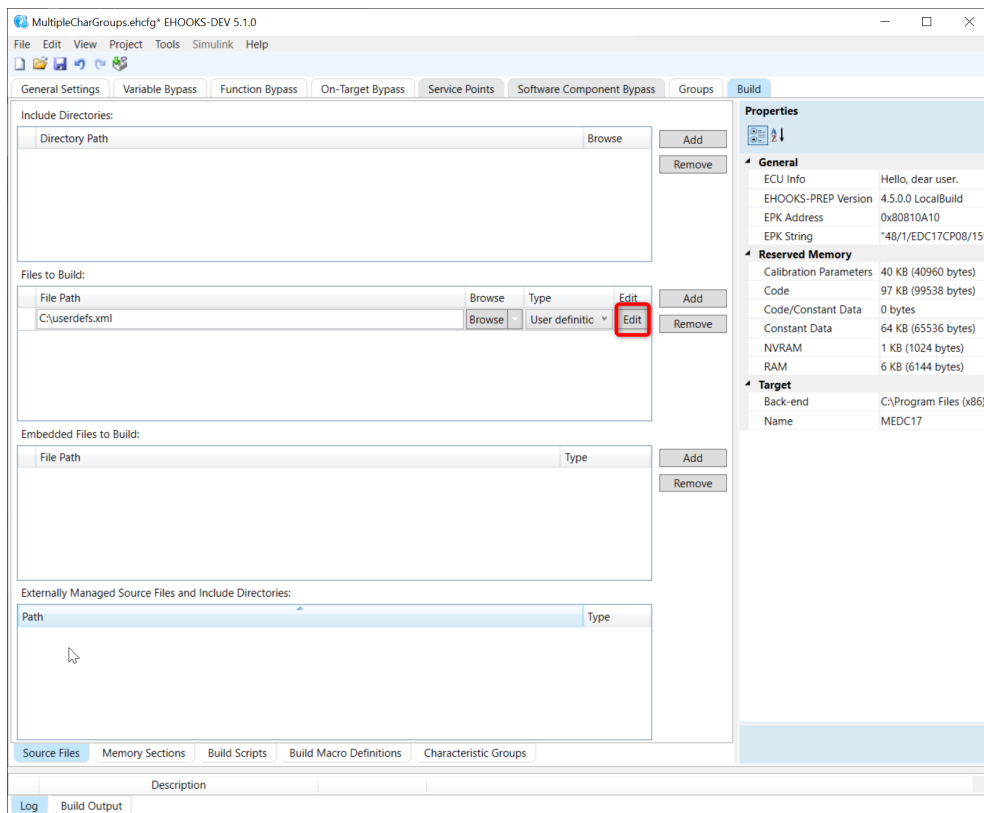


Figure 9.1: Launching the User Definition File Editor

The user definition file editor will then launch in a modal window as shown in figure 9.2. Each tab in the user definition editor allows a different object type to be created/edited.

- **Measurements:** Allows new scalar measurements to be introduced for measuring intermediate values of an on-target bypass algorithm
  - **Name:** The name of the measurement variable
  - **Description:** Textual description of the measurement. Will be added to the A2L file as a Long Identifier
  - **Type:** The A2L data type of the measurement variable
  - **Conversion Method:** The name of the A2L COMPU\_METHOD for the measurement variable
  - **Minimum:** The minimum value
  - **Maximum:** The maximum value
  - **Bit Mask:** The bit mask to be applied to the underlying value when displayed in a measurement/calibration tool such as INCA
  - **Create?:** Determines if EHOOKS should create the measurement implementation. If this is not specified then the on-target bypass code should create the implementation of the measurement as a global variable.

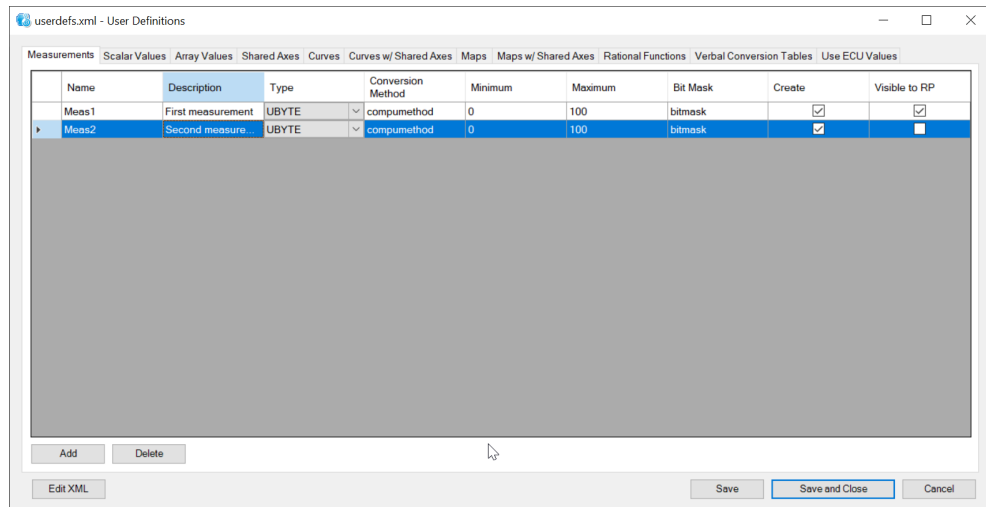


Figure 9.2: User Definition Editor

- Visible to RP?: Determines if EHOOKS will add an ASAP1b annotation to the measurement to enable it to be bypassed by an external rapid prototyping system. This can be useful, for example, to allow an external rapid prototyping system to be used to add a new hardware sensor and provide its value into an on-target bypass algorithm.

**NOTICE**

*EHOOKS will not add the code to update the measurement value with the value from the rapid prototyping system. Therefore, this code must be manually added to the on-target bypass code. This can be done by reusing the functions EHOOKS creates in the header files Characteristics.h and Tier1ExternalBypass.h. The functions EH\_GetExt<type> (handle, default) can be called. Where <type> is Byte, Word, Long or Float32. The handle is the EHOOKS-created ASAP1b channel/vector structure, which will always be named EH\_chanvec\_<name>, where name is the measurement variable name and default is the value that should be returned if the rapid prototyping system reports it is not properly connected, i.e. the default value. So, for example, to update a 16-bit measurement called Inject\_Manifold\_Pressure:*

```
Inject_Manifold_Pressure =
```

```
EH_GetExtWord(EH_chanvec_Inject_Manifold_Pressure, 0);
```

- Scalar Values: Allows new scalar calibration parameters to be created, enabling the calibration of an on-target bypass algorithm
  - Name: The name of the calibration parameter
  - Description: Textual description of the calibration parameter. Will be added to the A2L file as a Long Identifier
  - Type: The A2L data type of the calibration parameter
  - Conversion Method: The name of the A2L COMPU\_METHOD for the calibration parameter



- Minimum: The minimum value
- Maximum: The maximum value
- Bit Mask: The bit mask to be applied to the underlying value when displayed in a measurement/calibration tool such as INCA
- Initial Value: The initial physical value to be used with the reference page of the ECU software created by EHOOKS for the calibration parameter
- Array Values: Allows new arrays of calibration parameters to be created, enabling the calibration of an on-target bypass algorithm

As per Scalar Values, but with the addition of:

- Array Size: The number of elements in the array of calibration parameters
- Initial Values: The initial physical values for each array element. Within the table editor, the initial values can all be quickly set to a common default value. To set each initial value separately, click the **Edit XML** button and change the values directly in the XML file.
- Curves: Allows new curves to be created, enabling the calibration of an on-target bypass algorithm
  - Name: The name of the curve
  - Description: Textual description of the curve. Will be added to the A2L file as a Long Identifier
  - Format: Defines the layout format of the curve data structure in memory. EHOOKS supports the use of both ASCET and Simulink formats for maps and curves.
  - X-Axis Number of Points: The number of entries on the curve's X-axis
  - X-Axis Type for the Number of Points: The A2L data type for the number of data points on the curve's X-axis
  - X-Axis: Data Point Type: The A2L data type for the curve's X-axis values
  - X-Axis Conversion Method: The A2L COMPU\_METHOD for the curve's X-axis data points
  - X-Axis Minimum: The minimum value for the curve's X-axis data points
  - X-Axis Maximum: The maximum value for the curve's X-axis data points
  - Input Quantity: If a measurement is used as the input to the curve, then the measurement name can be given here. This is an optional element and is typically blank.
  - X-Axis Initial Values: The physical initial values for the curve's X-axis data points. Within the table editor, the initial values can all be quickly set to a common default value. To set each initial value separately, click the **Edit XML** button and change the values directly in the XML file.
  - Data Type: The A2L data type for the curve's data entries
  - Data Conversion Method: The A2L COMPU\_METHOD for the curve's data entries

- Data Minimum: The minimum value for the curve's data entries
- Data Maximum: The maximum value for the curve's data entries
- Data Initial Values: The physical initial values for the curve's data entries. Within the table editor, the initial values can all be quickly set to a common default value. To set each initial value separately, click the **Edit XML** button and change the values directly in the XML file.
- Maps: Allows new maps to be created, enabling the calibration of an on-target bypass algorithm

As per curves, but with the addition of Y-axis entries to correspond to the curve's X-axis entries.

- Rational Functions: Allows new A2L COMPU\_METHODS (type conversion functions / quantization formulae) to be defined
  - Name: The name of the new COMPU\_METHOD
  - Description: Textual description of the rational function. Will be added to the A2L file as a Long Identifier
  - Format: The A2L data format, indicating how many digits should be displayed
  - Unit: The physical unit string to be displayed in a measurement/calibration tool such as INCA
  - Coefficient A-F: The specification of the A2L conversion formulae using the standard A2L approach
- Verbal Conversion Tables: Allows new A2L TAB\_VERB COMPU\_METHODS to be defined
  - Name: The name of the new TAB\_VERB COMPU\_METHOD
  - Description: Textual description of the verbal conversion table. Will be added to the A2L file as a Long Identifier
  - Format: The A2L data format, indicating how many digits should be displayed
  - Unit: The physical unit string to be displayed in a measurement/calibration tool such as INCA
  - Pairs: The table of how to associate ECU values with descriptive labels
  - Default Value: The value to be displayed when the ECU value doesn't match any values specified in the Pairs list.
- Use ECU Values: Defines existing ECU scalar calibration parameter which can then be reused and accessed enabling calibration of an on-target bypass algorithm
  - Name: The name of an existing ECU scalar calibration parameter to be made available for reuse by on-target bypass code.

### 9.5.2 Extending the Example to Include a User Definition File

The example will be extended to introduce two scalar calibration characteristics, User\_Char\_1 and User\_Char\_2, and two scalar measurements, User\_Measurement\_1 and User\_Measurement\_2.

```

<OEMUserDefines xmlns="http://www.etas.com/EHOOKS/1.0/OEMUserDefines">
  <Measurement>
    <Name>User_Measurement_1</Name>
    <Type>ULONG</Type>
    <CompuMethod>OneToOne</CompuMethod>
    <Minimum>0</Minimum>
    <Maximum>4.2949673E+09</Maximum>
  </Measurement>
  <Measurement>
    <Name>User_Measurement_2</Name>
    <Type>ULONG</Type>
    <CompuMethod>OneToOne</CompuMethod>
    <Minimum>0</Minimum>
    <Maximum>4.294967E+09</Maximum>
    <Create />
  </Measurement>
  <ValueParameter>
    <Name>User_Char_1</Name>
    <Type>ULONG</Type>
    <CompuMethod>OneToOne</CompuMethod>
    <Minimum>0</Minimum>
    <Maximum>4.2949673E+09</Maximum>
    <InitialValues>
      <Value>1</Value>
    </InitialValues>
  </ValueParameter>
  <ValueParameter>
    <Name>User_Char_2</Name>
    <Type>ULONG</Type>
    <CompuMethod>OneToOne</CompuMethod>
    <Minimum>0</Minimum>
    <Maximum>4.294967E+09</Maximum>
    <InitialValues>
      <Value>1</Value>
    </InitialValues>
  </ValueParameter>
</OEMUserDefines>

```

**NOTICE**


---

The measurement `User_Measurement_2` uses the `<Create/>` element. This informs EHOOKS that the measurement variable associated with `User_Measurement_2` should be created by EHOOKS-DEV itself. If this element is not present then it is necessary for the on-target bypass code to provide the implementation of the measurement variable as a global variable.

---

The example on-target bypass code could then be updated to make use of these measurements and calibration characteristics as shown here:

#### On-Target Bypass Function with Measurement and Calibration Parameters

```

#include "ConversionFuncs.h"
#include "UserBypassFuncs.h"
#include "Characteristics.h"

float User_Measurement_1; /* User_Measurement_2 is created by EHOOKS */

```

```

EH_USER_BYPASS_FUNC(MyOTBFunc)
{
    /* Convert the input arguments into physical form. */
    float local_AccPed_stNSetP =
    EH_IMPL_TO_float_PHYS_AccPed_stNSetP(EH_ARG_GET_AccPed_stNSetP(EH_context));

    float local_AirCtl_mDesBasEOM0 =
    EH_IMPL_TO_float_PHYS_AirCtl_mDesBasEOM0(EH_ARG_GET_AirCtl_mDesBasEOM0(EH_context));

    User_Measurement_1 = (EH_ULONG) (local_AccPed_stNSetP +
                                    local_AirCtl_mDesBasEOM0);
    User_Measurement_2 = (EH_ULONG) (local_AirCtl_mDesBasEOM0 -
                                    local_AccPed_stNSetP);

    /* Write the output values in internal form. */
    EH_ARG_PUT_AC_trqDes(User_Measurement_1 + User_Char_1);
    EH_ARG_PUT_InjCrv_qPiI1Des__0__(User_Measurement_2 + User_Char_2);

    /* Bypass values valid for user */
    return 1;
}

```

No special implementation details are required within the on-target bypass code to make use of the measurements and calibration characteristics. EHOOKS provides the necessary macros and implementation details within the Characteristics.h header file.

---

**NOTICE**

*If map/curve calibration characteristics are created, the on-target bypass code must provide the necessary interpolation functions to perform the map/curve look-up functions. The ECU software provider may optionally provide an embedded header-file as part of the EHOOKS preparation of the ECU software to enable access to the interpolation routines built into the ECU software from within the on-target bypass code. If this is the case, details will be provided by the ECU software provider.*

---

## 9.6 Steps 5: Build and Run the EHOOKS-Created On-Target Bypass Software

Once the on-target bypass function has been configured and implemented, EHOOKS can be used to implement the hooks in the original ECU software, compile the user provided on-target bypass functions, and create the new measurements and calibration characteristics. The resulting HEX/s19 and a2l file can then be flashed to the ECU. The result of running an experiment in INCA is shown in figure 9.3.

## 9.7 EHOOKS On-Target Bypass Global Output Buffer Measurements

When on-target bypass is used in EHOOKS, a global output buffer is created to store the results coming from the on-target bypass functions. EHOOKS makes this buffer available as measurement values in the created hooked A2L file. For each variable hooked for on-target bypass (whether the on-target bypass is created manually, in Simulink or in ASCET), a measurement variable of the form EH\_gbvar\_<name> will be created, where is the name of the hooked ECU variable.

Using the on-target bypass global output buffer measurements can be very useful to enable the results of an on-target bypass function to be monitored before those bypass values are

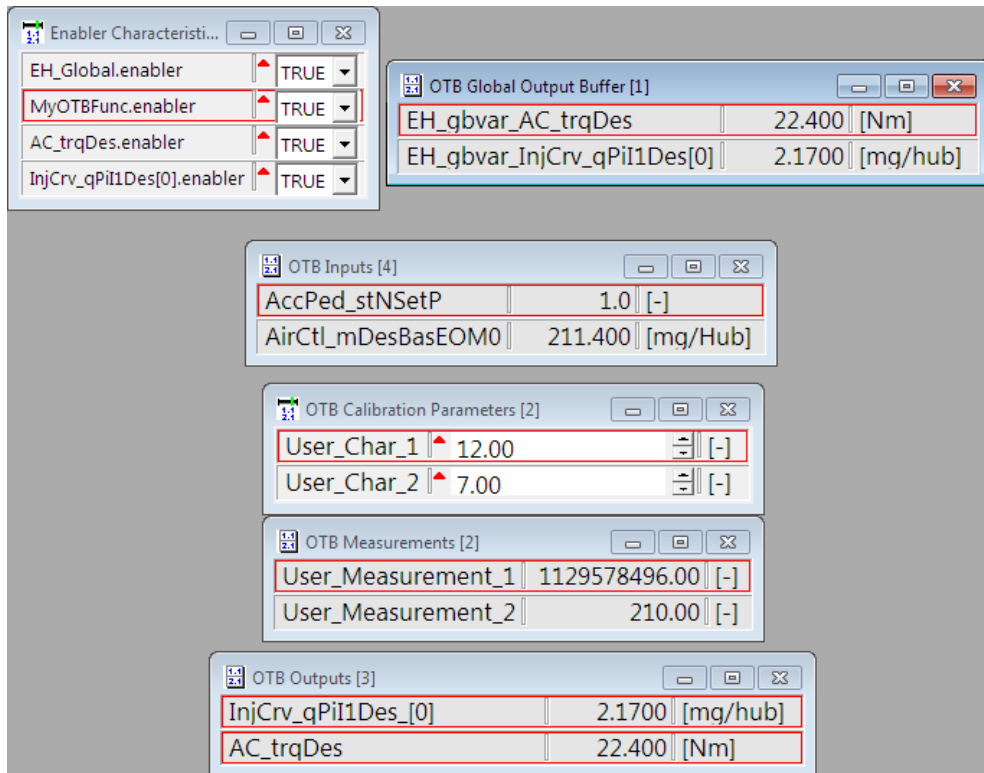


Figure 9.3: On-Target Bypass Experiment

written to the ECU variables. If the hooked ECU variable enablers are disabled, the output values from the on-target bypass will not be used, but they can still be monitored using these measurements. Figure 9.3 shows an example of measuring the on-target bypass global output buffer.

## 10 Creating and Working with On-Target Bypass from Simulink

### 10.1 Introduction

The EHOOKS-DEV Simulink Integration package makes it efficient and easy to configure EHOOKS and integrate Simulink models for on-target bypass. Using the EHOOKS-DEV Simulink Integration package adds new Simulink blocks and system targets that allow the entire EHOOKS configuration and build process to be managed directly from within Simulink. This section details how to use the EHOOKS-DEV Simulink Integration package to create an EHOOKS on-target bypass configuration, how to integrate EHOOKS blocks with your Simulink model and how to build the hooked ECU software. While the focus of this section is on Simulink model-based on-target prototyping, the EHOOKS-DEV Simulink Integration package gives access to the full EHOOKS environment and therefore all EHOOKS features can be used. It is, therefore, perfectly possible to create configurations mixing Simulink based on-target bypass with constant, calibration, external and manual C code external bypass using the EHOOKS-DEV Simulink Integration package.

---

**NOTICE**

*The EHOOKS Simulink blockset is designed to support on-target-bypass experiments only. The Simulink blockset should not be used for simulation experiments as the behavior of the blocks is not neutral in a simulation environment and will therefore lead to unexpected results.*

---

### 10.2 EHOOKS Blocks for Simulink

The EHOOKS-DEV Simulink Integration package introduces a number of Simulink blocks to be used both to configure EHOOKS and to establish the connection between the ECU software and the Simulink model.

When the EHOOKS-DEV Simulink Integration package is installed and configured, an EHOOKS block library folder will appear within the Simulink Library Browser. Within the EHOOKS folder the main EHOOKS configuration block can be found. The EHOOKS built-in block folder contains some standard Simulink blocks pre-configured for use with the EHOOKS module. The EHOOKS library blocks folder shown in figure 10.1 contains the EHOOKS block-set that enables the configuration of EHOOKS and the integration of the ECU software and the Simulink model.

In the following sections each of the EHOOKS blocks for Simulink will be introduced and the functionality outlined. For the details of how to use these blocks within Simulink for on-target bypass modelling see section [Simulink Modelling for On-Target Bypass](#).

---

**NOTICE**

*EHOOKS V4.9 introduced an upgraded Simulink blockset. Existing Simulink models that include blocks created with earlier versions are automatically migrated to use the new EHOOKS V4.9 blockset. It should be noted however that Simulink models created with EHOOKS V4.9 blocks cannot be used with earlier Simulink blocksets.*

---

#### 10.2.1 EHOOKS Configuration Block

The EHOOKS configuration block (see figure 10.2) adds EHOOKS support to a Simulink model. This block must be added to the top level of the Simulink model and must be configured before attempting to work with the other EHOOKS blocks.

To manage the EHOOKS configuration represented by the EHOOKS configuration block, simply

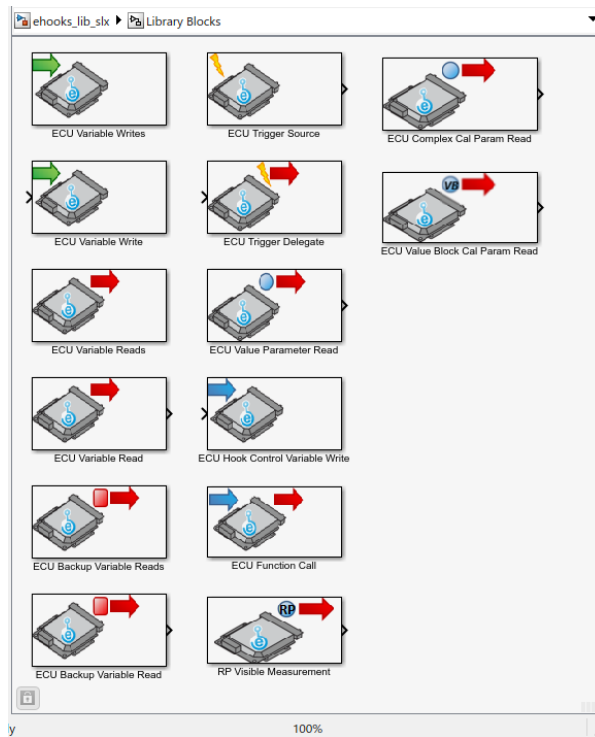


Figure 10.1: EHOOKS Blockset for Simulink On-Target Bypass

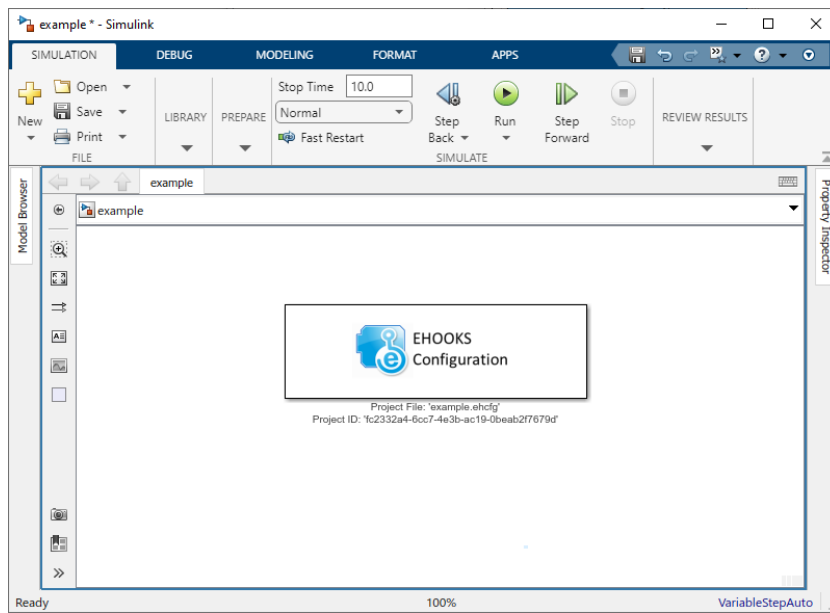


Figure 10.2: EHOOKS Configuration Block

**double-click** it. This will launch an initial Simulink interface (see figure 10.3) to manage which EHOOKS configuration file is associated with the configuration block.

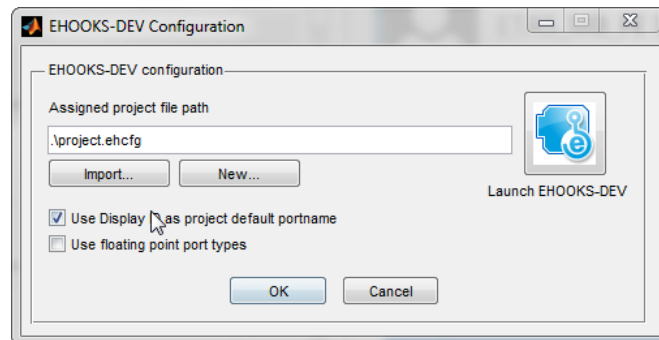


Figure 10.3: EHOOKS Configuration Block Simulink Interface

The EHOOKS-DEV configuration section allows the Simulink model (and the EHOOKS configuration block) to be associated with an EHOOKS configuration file.

- To use an existing EHOOKS configuration file, click the **Import** button to locate the EHOOKS configuration file (or simply type the file name/path into the text field) and then click **OK**. A relative path to the EHOOKS configuration file will be generated if possible.
- To create a new EHOOKS configuration file, simply click the **New** button to define the new file name and location, and then click **OK**.

---

**NOTICE**

*It is recommended that the EHOOKS configuration file is created in the same directory as the Simulink model (default behaviour) to allow the model directory to be easily copied from one computer to another without needing to reconfigure the path names. The base directory for relative paths to the EHOOKS configuration file is always the model directory.*

*Please ensure that the model file can always be located by MATLAB, either by changing the current working directory or by updating the MATLAB search path.*

---

Once an EHOOKS configuration file has been assigned to the EHOOKS configuration block, its path is displayed in the Simulink model directly below the EHOOKS configuration block as shown in figure 10.2 along with the EHOOKS project identifier. Creating a new EHOOKS configuration or opening the existing EHOOKS configuration will launch the main EHOOKS-DEV front-end (see figure 5.1, in section 5 Configuring EHOOKS-DEV). Once the EHOOKS-DEV front-end is loaded, it is recommended to simply minimize it rather than close it to make interaction with the other EHOOKS Simulink blocks more efficient.

---

**NOTICE**

*If the EHOOKS-DEV front-end is not opened then, when an EHOOKS Simulink block is configured, the EHOOKS-DEV front-end will first have to be launched, taking several seconds. To prevent this, it is recommended to simply keep the EHOOKS-DEV front-end opened in the background (or minimized) all the time when working with a Simulink model using EHOOKS blocks.*

---



---

**NOTICE**

*When EHOOKS-DEV (or one of its dialog boxes) is launched from within Simulink by double-clicking on a Simulink block, sometimes the EHOOKS-DEV window will not always come to the front. Therefore, it may be hidden from view behind the Simulink window. It is therefore necessary to check behind the Simulink window to find the EHOOKS-DEV dialogs.*

*When an EHOOKS-DEV dialog is present on the screen, the Simulink user interface may become unresponsive (the actual behaviour depends on the used MATLAB version). Simply close the EHOOKS-DEV dialog to return control to Simulink to continue working.*

---

Two additional checkboxes are provided to complete the configuration process:

- The **Use Display ID as project default portname** checkbox causes all ECU variables displayed by the various parts of the EHOOKS Simulink Integration package to display by default the A2L DISPLAY\_IDENTIFIER for the variable rather than the A2L ECU variable name. The usage of display identifiers can then still be disabled on a per-block basis.
- The **Use Floating point port types** checkbox causes all ECU variable/parameter read/write block ports to use the floating point data type (i.e. data type *double*) if signal conversion is turned off. If the option is disabled, the original A2L variable/parameter data-type is used for ECU variable/parameter read/write block ports if signal conversion is turned off. For a Simulink model created with the EHOOKS V3.0 blockset (or later) the option is disabled by default. When a Simulink model created with the EHOOKS V2.0 blockset is imported, the option is enabled by default to ensure backwards compatibility. This checkbox therefore allows the user to choose between using EHOOKS V2.0 or V3.0 default behavior.

### 10.2.2 EHOOKS ECU Trigger Source Block

The EHOOKS ECU trigger source block (ECU trigger block), shown in figure 10.4, allows Simulink model code to be integrated into the execution of the ECU software.

The ECU trigger block is a function-call generator block that should be attached to a Simulink function-called triggered subsystem. An ECU trigger block effectively represents a single on-target bypass function to be integrated into the ECU software by EHOOKS.

---

**NOTICE**

*The name of the created on-target bypass function will match the name of the ECU trigger block (the name will be made C conformant though). If a specific on-target bypass function name is desired, the block should be renamed to contain this name.*

---



---

**NOTICE**

*The ECU trigger block only provides a scalar trigger signal. If multiple trigger sources for the same on-target bypass function are required then a de-multiplexer block can be used to create them via forking out the signal.*

---

To configure the ECU trigger block, simply **double-click** it. This will launch the EHOOKS configuration interface for the ECU trigger block, as shown in figure 10.5. The **Dispatch Point**

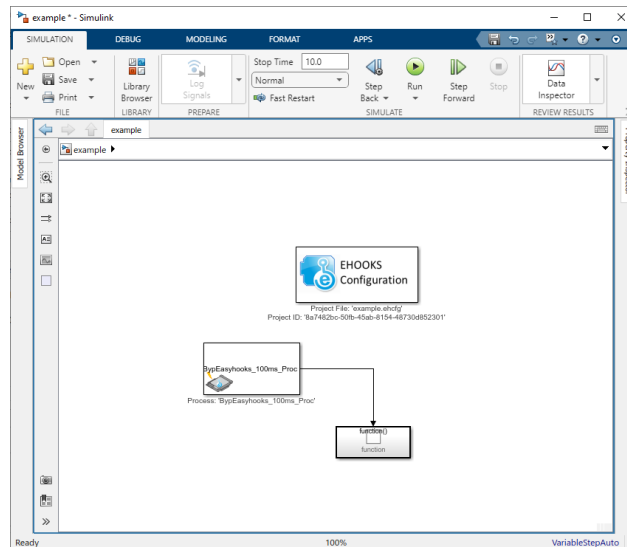


Figure 10.4: EHOOKS ECU Trigger Source block

section of the dialog allows the selection of an existing ECU process or the EHOOKS hook (of an associated measurement variable) to be selected to trigger the execution of the on-target bypass function. The list displays the set of ECU processes that have been prepared by the ECU software provider for the purposes of dispatching on-target bypass functions, and the set of hooked ECU measurement variables. If the list is very long, then it can be filtered using the filter text box. The filter string can include the wildcard characters ? and \*. ? will match any single character and \* will match any number of characters. Note that the filter string contains an implicit \* wildcard at the end of the string. The character \$ can be used to match the end of a variable name. As the filter string is updated, the variable list will dynamically update to show any matching ECU process.

In some situations, the ECU software provider may have prepared some ECU processes to allow their use to dispatch on-target bypass functions, but indicated that their names should not be displayed in the EHOOKS-DEV user interface. In this case, the bypass container can be added to the configuration by typing its name into the **Manual Add** text field and clicking the **Add** button.

The **Dispatch Type** drop-down allows configuration of whether the Simulink model code will be executed before (Pre-Dispatch, default) or after (Post-Dispatch) the bypass container

By default, hooked variables are updated with the current bypass value at the point where the ECU variable would have originally been written by the delivered ECU software. Additionally, by checking the **Force Writes to Outputs at Dispatch Point** checkbox, EHOOKS enables the configuration of forced-writes directly after the on-target bypass function has executed. When this option is configured, the calculated bypass values will be immediately and forcibly written into the ECU measurement variables configured as outputs to the on-target bypass function.

Elevated Permissions may be enabled from the configuration interface, by checking **Force Writes to Output Dispatch Point** followed by **Elevated Permissions for Force Writes**. To enable the checkbox, the **Enable Elevated Permissions** project option must be enabled from EHOOKS-DEV Front-End (see [11.5.4 Enable Elevated Permissions](#)).

The **General Properties** section allows the use of an **Enabler** and **Indicator** to be

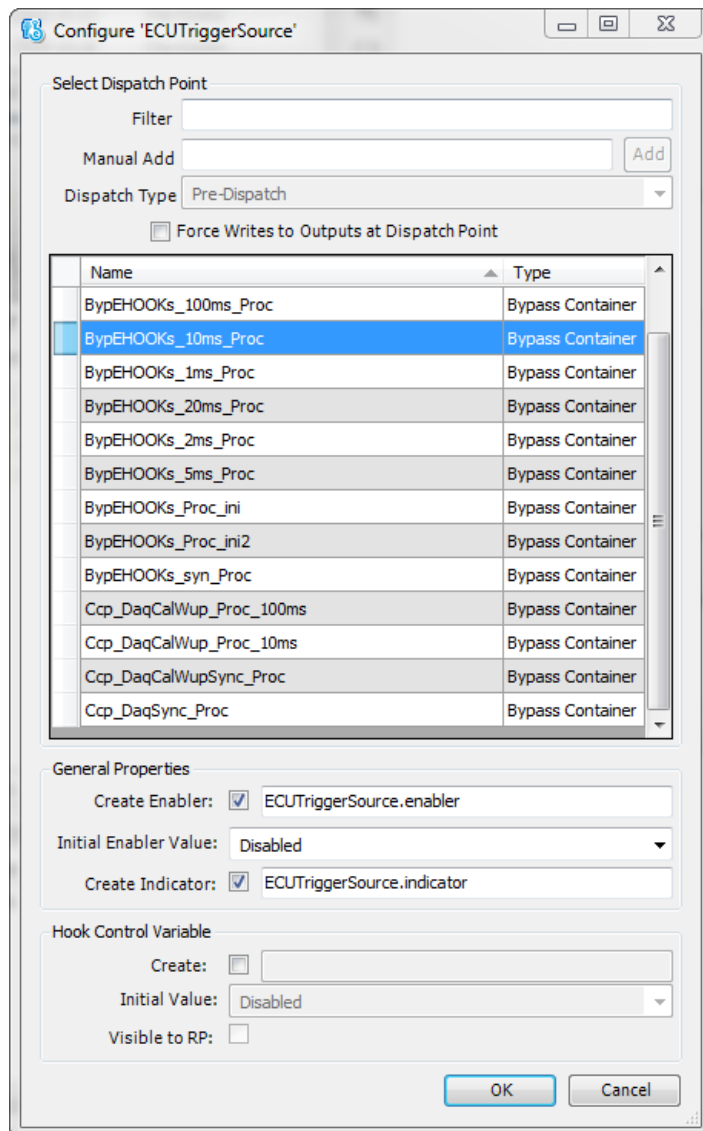


Figure 10.5: EHOOKS ECU Trigger Source Configuration

configured, and displays a read-only list of the groups which include this ECU trigger block. The **Enabler** settings allow the configuration of whether EHOOKS will add a calibration characteristic that can then be used at run-time to control whether or not the on-target bypass function is activated. To configure an enabler, the checkbox must be ticked and then the name of the enabler calibration characteristic can be set in the text field. If an enabler is configured, it is also possible to configure an indicator. The **Indicator** setting allows the configuration of whether EHOOKS will add a measurement variable that will mirror the current value of the enabler characteristic at run-time. To configure an indicator, the checkbox must be ticked and then the name of the indicator measurement variable can be set in the text field.

---

**NOTICE**

*To deselect a dispatch point, press the Ctrl button while selecting the highlighted item. An undefined dispatch point is particularly useful if the OTB function shall be the subject of manual dispatching (see ECU trigger delegation block (see section [10.2.6 EHOOKS ECU Trigger Delegate Block](#))).*

---

Finally, a hook control variable can be added to the configuration by ticking the **Create** checkbox in the Hook Control Variable section and setting the name to be used for the control variable. This control variable can then be managed at run-time to determine if the on-target bypass function is enabled or disabled. The control variable can be managed via a C-code on-target bypass function, or by the Simulink model using the ECU hook control variable write block (see section [10.2.7 EHOOKS ECU Hook Control Variable Write Block](#)).

The **Initial Value** of the hook control variable can be set to either Disabled (default) or Enabled. Setting the **Initial Value** to Enabled allows the hook to be enabled when the ECU is first powered up.

Finally, the hook control variable can be configured to **Visible to RP**. If this option is configured, it is possible to use INTECRIO to write to the hook control variable from bypass code running on external rapid prototyping hardware.

---

**NOTICE**

*The configured control variable name will be created as a C variable that can then be managed at run-time by an on-target bypass function. The configured name should therefore be a valid C identifier. However, EHOOKS will automatically convert the name into a valid C identifier if the name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (\_). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.*

*The converted C name will need to be used in any on-target bypass function that wants to manage a control variable at run-time.*

---

Once configured, EHOOKS will ensure that the hooked ECU software will execute the model code contained within the attached function-called subsystem whenever the ECU process associated with the EHOOKS ECU function trigger source block is executed.

### 10.2.3 EHOOKS ECU Variable Reads Block

The EHOOKS ECU Variable Reads block, shown in figure [10.6](#), allows Simulink models to read from multiple existing ECU variables.

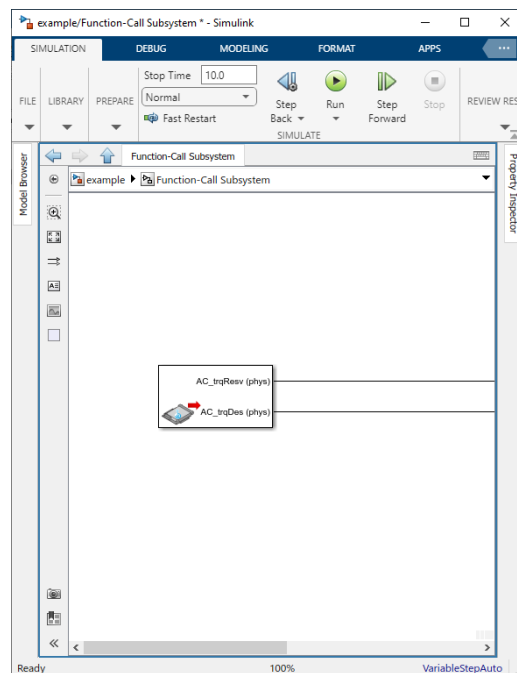


Figure 10.6: EHOOKS ECU Variable Reads Block

The ECU Variable Reads block is effectively a data source block that can be attached to the Simulink model to provide access to ECU data. An ECU Variable Reads block must only be placed inside a subsystem that is connected to an ECU trigger block; the ECU Variable Reads block can be used at any hierarchical level in the model as necessary provided this condition is maintained. The model can contain as many ECU Variable Reads blocks as required to help structure the model.

---

**NOTICE**

*Since the ECU variable read assignments are placed globally, the ECU Variable Reads blocks don't work with the code reuse concept of Simulink. If you plan to use code reuse subsystems (or similar constructs) please place the ECU Variable Reads blocks outside of the model parts that shall be subject of code reuse.*

---

To configure the ECU Variable Reads block, simply **double-click** it. This will launch the EHOOKS configuration interface for the ECU Variable Reads block, as shown in figure 10.7. The left-hand column allows the displayed variable list to be filtered by the A2L file function groups (including filtering by the input, output and local measurements to a function). To quickly move to the desired function group, click on an entry in the function group list to move focus and then begin typing.

The middle-column displays a variable list which can be filtered by typing a filter string into the **Filter String** text box. The filter string can include the wildcard characters ? and \*. ? will match any single character and \* will match any number of characters. Note that the filter string contains an implicit \* wildcard at the end of the string. The character \$ can be used to match the end of a variable name. As the filter string is updated, the variable list will dynamically update to show any matching ECU variables that can be read.

The right-hand column shows the ECU variables that have been selected for reading. ECU

variables can be moved in and out of this list using the >> and << buttons, or by pressing Ctrl+right-arrow and Ctrl+left-arrow, respectively. Additionally, **double-clicking** on an entry in either list will move it to the other list. If an array variable has been selected for reading, the elements of the array to be hooked can be configured by clicking on the ... button. The number of elements to be read is then shown in the right-hand column. For each variable, the **Convert** checkbox indicates whether the variable should be read in ECU implementation form or if it should first be converted to physical representation before being passed into the Simulink model. If the check box is ticked, the associated variable will be converted by EHOOKS to physical representation. The **Convert All** checkbox allows all selected variables to be quickly changed between being read in ECU implementation or physical representations.

Finally, the order in which the selected variables will be displayed with the ECU variable read block will match the order of the variables in the selected variable list. This can be adjusted by selecting a variable and using the and buttons.

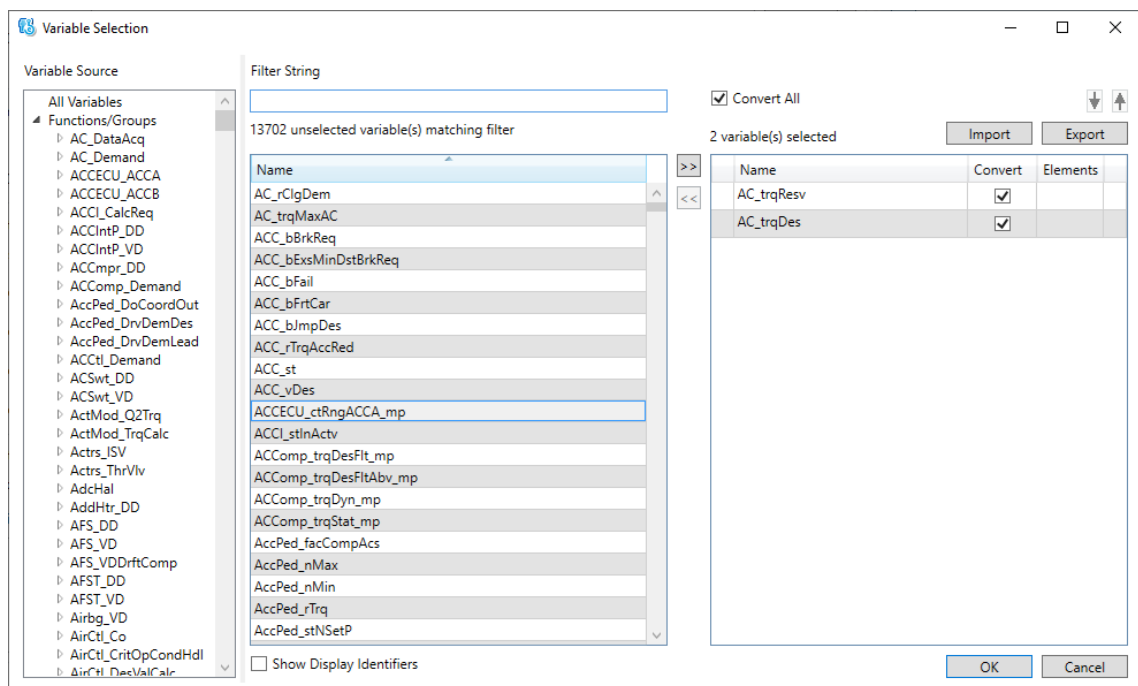


Figure 10.7: EHOOKS ECU Variable Reads Block Configuration

Once the **OK** button is selected, the ECU Variable Reads block will be updated to contain an output port for each selected variable. The name of each port will match the associated ECU variable name and will indicate whether the variable is being accessed in ECU implementation or physical representation.

#### NOTICE

*It is possible to read the same ECU variable in more than one ECU variable read block. This can be used to allow the same ECU variable to be read in both ECU implementation and physical representation.*

### 10.2.4 EHOOKS ECU Variable Writes Block

The EHOOKS ECU Variable Writes block, shown in figure 10.8, allows Simulink models to write to multiple existing ECU variables (i.e. write to ECU variables that EHOOKS will hook within the

ECU software).

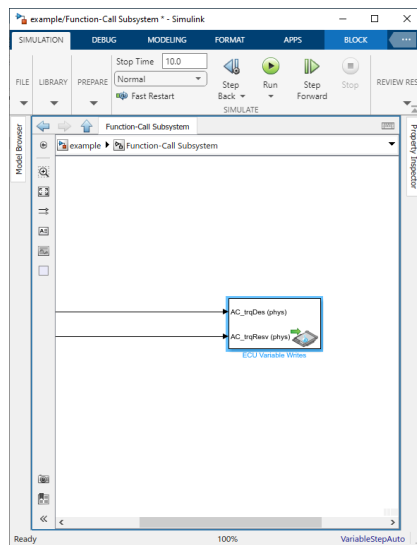


Figure 10.8: EHOOKS ECU Variable Writes Block

The EHOOKS ECU Variable Writes block is effectively a data sink block that can be attached to the Simulink model to allow it to bypass existing ECU variables. An EHOOKS ECU Variable Writes block must only be placed inside a subsystem that is connected to an ECU trigger block; the ECU Variable Writes block can be used at any hierarchical level in the model as necessary provided this condition is maintained. The model can contain as many ECU variable write blocks as required to help structure the model.

To configure the ECU Variable Writes block, simply **double-click** it. This will launch the EHOOKS configuration interface for the ECU Variable Writes block, as shown in figure 10.9. The left-hand column allows the displayed variable list to be filtered by the A2L file function groups (including filtering by the input, output and local measurements to a function). To quickly move to the desired function group, click on an entry in the function group list to move focus and then begin typing.

The middle-column displays a variable list which can be filtered by typing a filter string into the **Filter String** text box. The filter string can include the wildcard characters ? and \*. ? will match any single character and \* will match any number of characters. Note that the filter string contains an implicit \* wildcard at the end of the string. The character \$ can be used to match the end of a variable name. As the filter string is updated, the variable list will dynamically update to show any matching ECU variables that can be hooked.

The right-hand column shows the ECU variables that have been selected for being written to. ECU variables can be moved in and out of this list using the >> and << buttons, or by pressing Ctrl+right-arrow and Ctrl+left-arrow, respectively. Additionally, **double-clicking** on an entry in either list will move it to the other list. If an array variable has been selected for being written to, the elements of the array to be written can be configured by clicking on the ... button. The number of elements to be written to is then shown in the right-hand column. For each variable, the **Convert** checkbox indicates whether the variable coming from the Simulink model is in ECU implementation form or if it is in physical representation and EHOOKS should convert it to implementation form before writing it back to the ECU variable. If the check box is ticked, the associated variable will be converted by EHOOKS from physical representation. The **Convert All** checkbox allows all selected variables to be quickly changed

between being provided in ECU implementation or physical representations.

Finally, the order in which the selected variables will be displayed with the ECU variable write block will match the order of the variables in the selected variable list. This can be adjusted by selecting a variable and using the ^ and v buttons.

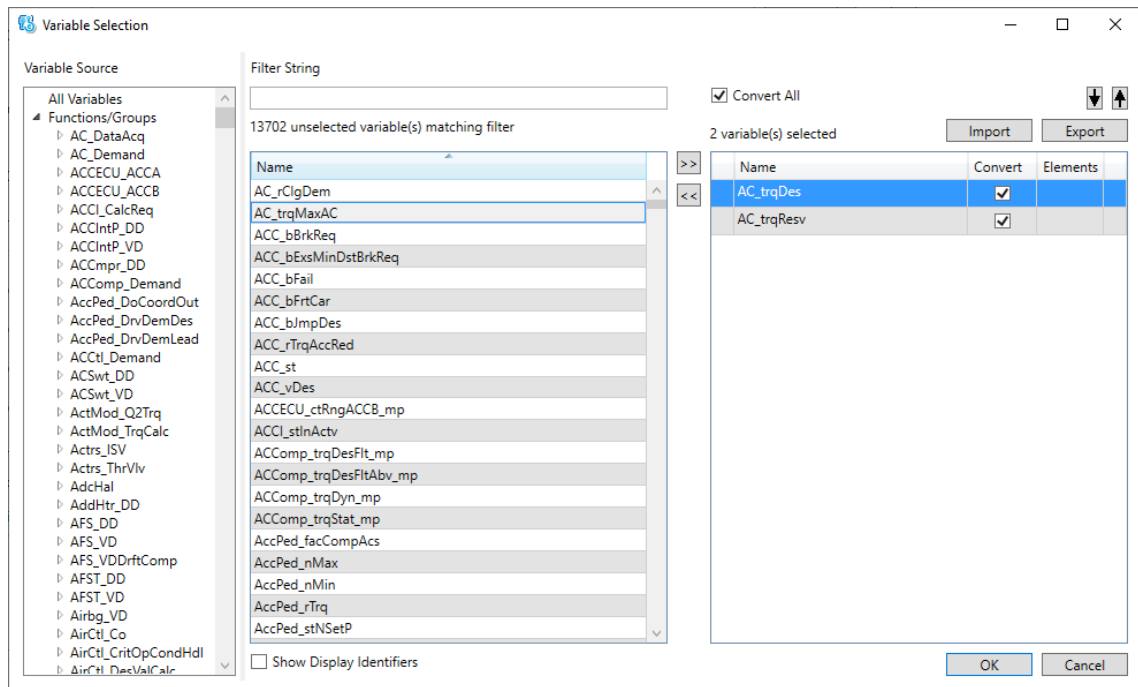


Figure 10.9: EHOOKS ECU Variable Writes Block Configuration

Once the **OK** button is selected, the ECU Variable Writes block will be updated to contain an input port for each selected variable. The name of each port will match the associated ECU variable name and will indicate whether the variable is being accessed in ECU implementation or physical representation.

#### NOTICE

*It is possible to write the same ECU variable in more than one ECU variable writes block. Typically this is useful if the on-target bypass experiment requires a different calculation for a variable depending on the ECU operation mode. Care should be taken if writing to the same ECU variable to avoid the case where the ECU variable flip-flops between the provided values. This can be achieved by ensuring the correct conditional code is introduced into the model or through the use of EHOOKS control variables to programmatically enable/disable on-target bypass functions (to ensure that only one of the on-target bypass functions that writes to a specific variables is enabled at any one time).*

Once an ECU variable has been added to an ECU Variable Writes block, EHOOKS will create a hook of type on-target bypass for this ECU variable. The additional properties of the variable hooks added via the ECU Variable Writes block can be configured in the main EHOOKS-DEV user interface. If EHOOKS-DEV is not already running, simply **double-click** on the EHOOKS configuration block to launch it. To configure the additional properties of a hook added via the ECU variable write block switch to the **Variable Bypass** tab, and then select the **Modeling**



**Tool** hooked variable list using the tab at the bottom of the window.

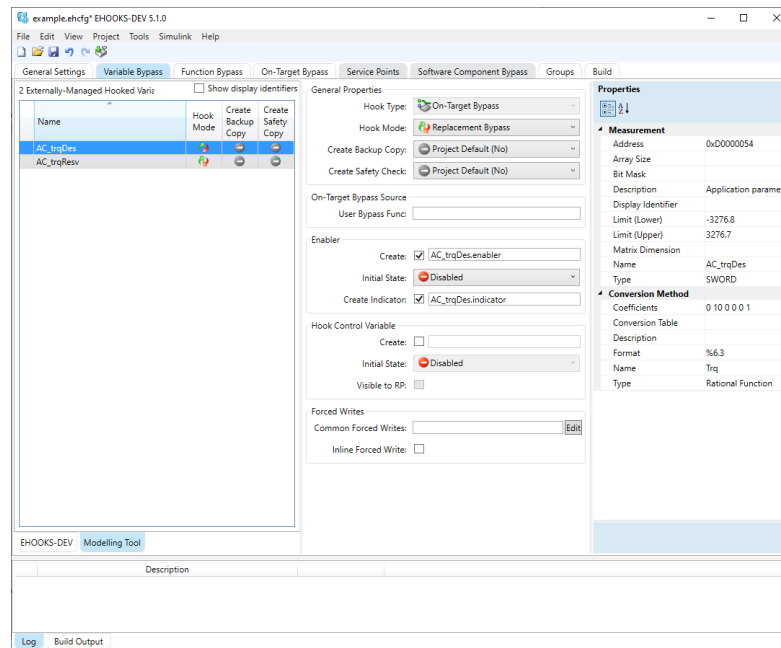


Figure 10.10: Configuring Hook Properties for an ECU Variable Writes Block

The interface for configuring the hook properties is identical to that for ECU variable hooks as described in section 5.2.2 [Configuring Properties of a Variable Hook](#), with the exception that the **Hook Type** cannot be modified from the selected **On-Target Bypass** type and hence there is no need to configure the corresponding **Source** section.

The **Hook Mode** drop down allows the way in which the bypass value is used by the hook to be configured. The Hook Mode can be:

- **Replacement Bypass**

The configured bypass value will be used by the hook to completely replace (overwrite) the ECU calculation for the hooked ECU variable.

- **Offset Bypass**

The configured bypass value will be added or subtracted to the ECU calculation for the hooked ECU variable.

- **Multiply Bypass**

The configured bypass value will be multiplied with the ECU calculation for the hooked ECU variable.

The **Create Backup Copy** drop down allows the configuration of whether EHOOKS will create new ECU variables to record a copy of the ECU calculation for the hooked ECU variables. The drop down allows the selection from three options:

- **Project Default**

The creation of the backup copy ECU measurements will be determined based on the project wide setting configured on the General Settings Tab (see section 5.1.3 [Project Settings](#)). This makes it quick and easy to change the setting for all the hooks together.

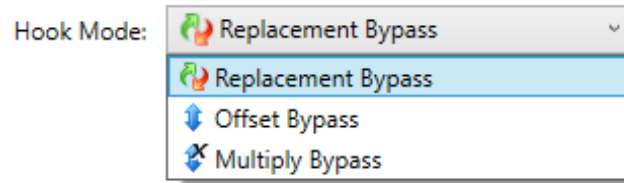


Figure 10.11: Configuring the Hook Mode

- **Yes**

This means EHOOKS will create a new measurement variable and keep it up to date with the most recent ECU calculation for the hooked ECU variable.

- **No**

This means EHOOKS will not create a backup copy measurement.

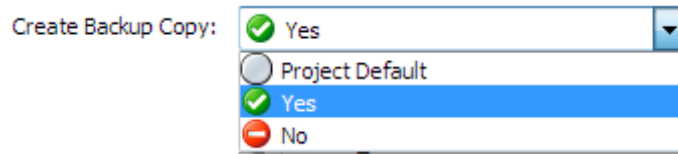


Figure 10.12: Configuring the Backup Copy Setting

The **Create Safety Check** drop down allows the configuration of whether EHOOKS will add additional run-time code to monitor the run-time performance of the hook. This code will determine if the hook is running correctly and, if not, will disable the hook and indicate that an error has been detected. The drop down allows section from three options:

- **Project Defaults**

The addition of safety-checking code will be determined based on the project wide setting configured on the General Settings Tab (see section [5.1.3 Project Settings](#)). This makes it quick and easy to change the setting for all the hooks together.

- **Yes**

This means EHOOKS will create safety checking code for the hooked ECU variable to monitor its performance at run-time.

- **No**

This means EHOOKS will not monitor the performance of the hooked ECU variable at run-time.

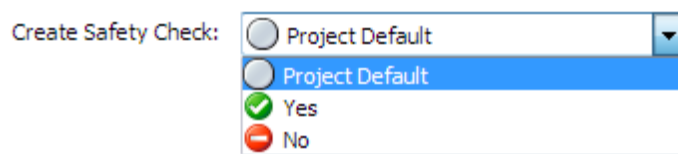


Figure 10.13: Configuring the Safety Checks Setting

The **Enabler** settings allow the configuration of whether EHOOKS will add a calibration

characteristic that can then be used at run-time to control whether or not the hook is activated. To configure an enabler, the checkbox must be ticked and then the name of the enabler calibration characteristic can be set in the text field. The initial value of the hook enabler is set to disabled by default, which will cause the hook to be turned off at ECU startup. The initial value of the hook enabler can be changed to enabled from the General Properties section, which will allow the hook to be turned on at ECU startup.

If an enabler is configured, it is also possible to configure an indicator. The **Indicator** setting allows the configuration of whether EHOOKS will add a measurement variable that will mirror the current value of the enabler characteristic at run-time. To configure an indicator, the checkbox must be ticked and then the name of the indicator measurement variable can be set in the text field.

The screenshot shows a configuration dialog with three rows. The first row is 'Create Enabler:' with a checked checkbox and a text field containing 'EH\_G314\_sbyte3.enabler'. The second row is 'Initial Enabler Value:' with a dropdown menu showing 'Disabled' and a red stop icon. The third row is 'Create Indicator:' with a checked checkbox and a text field containing 'EH\_G314\_sbyte3.indicator'.

Figure 10.14: Configuring Variable Bypass Enabler and Indicator Settings

In the **Advanced** section of the hook property configuration, there are three configuration options.

The screenshot shows an 'Advanced' section with three options. 'Write Locations:' has a text field with 'BypEasyhooks\_100ms\_Proc, BypEa:' and an 'Edit' button. 'Inline Forced Write:' has an unchecked checkbox. 'Control:' has a checked checkbox and a text field with 'AccPed\_stNSetP.control'.

Figure 10.15: Configuring Variable Bypass Advanced Settings

The first two of these relate to forcing the update of a bypass value for a hooked ECU variable. The **Write Locations** displays a read-only list of the configured ECU processes in which an update of the bypass value for the hooked ECU variable will be performed. The set of write locations can be updated by clicking the **Edit** button. This will display the **Select Forced Write Locations** dialog. In the left column, a list of the available ECU processes is displayed. EHOOKS can be configured to force the update of the bypass value in one or more of these processes by moving them into the right column by first selecting them and then using the >> button. To remove a forced write location from the configuration, first select it and click the << button. The dialog offers a filter to help locate the desired ECU process; this behaves in the same way as for variable selection (see section 5.2.1 [Selecting Variables to Be Hooked](#)).

In some situations, the ECU software provider may have prepared some ECU processes to allow their use as forced write location, but indicated that their names should not be displayed in the EHOOKS-DEV user interface. In this case, such forced write locations can be added to the configuration by typing their name into the **Manual Add** text field and clicking the **Add** button.

The **Inline Forced Write** check box allows the configuration of a forced write that EHOOKS will automatically insert at the start of every ECU process where the hooked ECU variable is updated within the original ECU software.

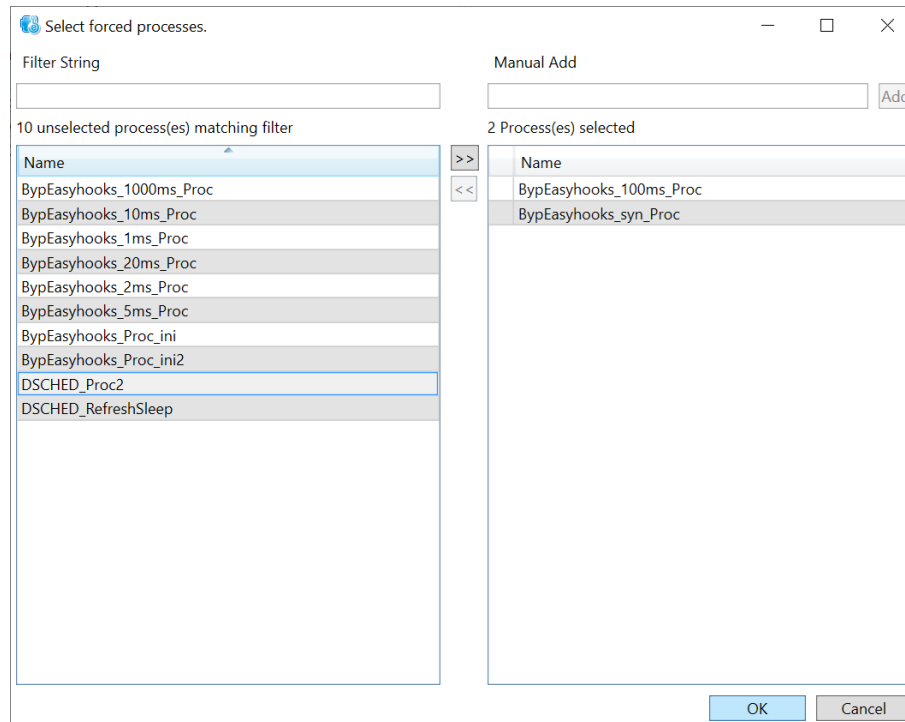


Figure 10.16: Selecting Forced Write Locations

Finally, a control variable can be added to the configuration by ticking the **Control** checkbox and setting the name to be used for the control variable.

---

**NOTICE**

The configured control variable name will be created as a C variable that can then be managed at run-time by an on-target bypass function. Therefore the configured name should be a valid C identifier. However, EHOOKS will automatically convert the name into a valid C identifier if the name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (`__`). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.

The converted C name will need to be used in any on-target bypass function that wants to manage a control variable at run-time.

---

### 10.2.5 EHOOKS ECU Backup Variable Reads Block

The EHOOKS ECU Backup Variable Reads block, shown in figure 10.17, allows Simulink to read from the ECU calculated values for multiple hooked ECU variables (i.e. the values calculated by the original ECU software before the ECU variables were hooked and bypassed by EHOOKS).

The ECU Backup Variable Reads block is effectively a data source block that can be attached to the Simulink model to provide access to ECU data. An ECU Backup Variable Reads block must only be placed inside a subsystem that is connected to an ECU trigger block; the ECU Backup Variable Reads block can be used at any hierarchical level in the model as necessary, provided this condition is maintained. The model can contain as many ECU backup variable read blocks as required to help structure the model.

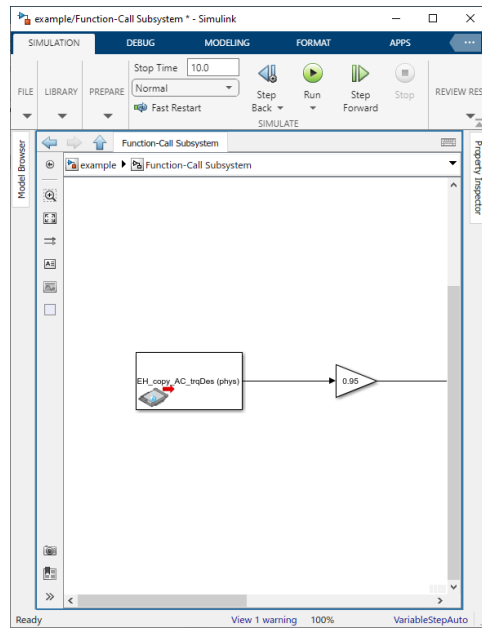


Figure 10.17: EHOOKS ECU Backup Variable Reads Block

To configure the ECU Backup Variable Reads block, simply **double-click** it. This will launch the EHOOKS configuration interface for the ECU Backup Variable Reads block, as shown in figure 10.18. The left-hand column allows the displayed variable list to be filtered by the A2L file function groups (including filtering by the input, output and local measurements to a function). To quickly move to the desired function group, click on an entry in the function group list to move focus and then begin typing.

The middle-column displays a variable list which can be filtered by typing a filter string into the **Filter String** text box. The filter string can include the wildcard characters ? and \*. ? will match any single character and \* will match any number of characters. Note that the filter string contains an implicit \* wildcard at the end of the string. The character \$ can be used to match the end of a variable name. As the filter string is updated, the variable list will dynamically update to show any matching ECU variables that have backup variables that can be selected for reading.

The right-hand column shows the EHOOKS-created backup variables that have been selected for reading. Backup variables can be moved in and out of this list using the >> and << buttons respectively. Additionally, **double-clicking** on an entry in either list will move it to the other list. For each backup variable, the **Convert** checkbox indicates whether the variable should be read in ECU implementation form or if it should first be converted to physical representation before being passed into the Simulink model. If the check box is ticked, the associated variable will be converted by EHOOKS to physical representation. The **Convert All** checkbox allows all selected variables to be quickly changed between being read in ECU implementation or physical representations.

Finally, the order in which the selected variables will be displayed with the ECU Backup Variable Reads block will match the order of the variables in the selected variable list. This can be adjusted by selecting a variable and using the ^ and v buttons.

Once the **OK** button is selected, the ECU Backup Variable Reads block will be updated to contain an output port for each selected variable. The name of each port will match the

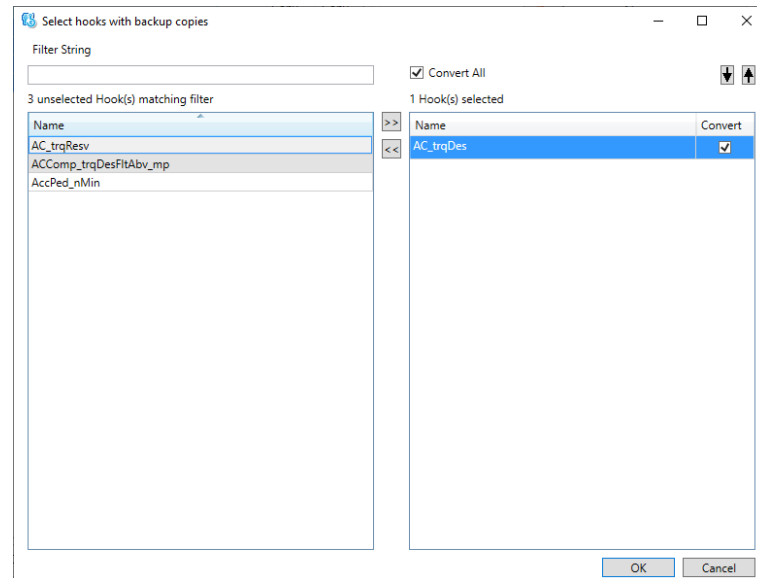


Figure 10.18: EHOOKS ECU Backup Variable Reads Block Configuration

associated ECU variable name and will indicate whether the variable is being accessed in ECU implementation or physical representation.

---

**NOTICE**

*It is possible to read the same backup measurement in more than one ECU Backup Variable Reads block. This can be used to allow the same backup variable to be read in both ECU implementation and physical representation.*

---

### 10.2.6 EHOOKS ECU Trigger Delegate Block

The EHOOKS ECU Trigger Delegate Block, shown in figure 10.19, allows a Simulink model to trigger the execution of an EHOOKS on-target bypass function.

The ECU Trigger Delegate Block is effectively a function-call sink and should be connected to a function-call generator. The source of the connected function-call generator must be contained within a subsystem hierarchy that is connected to an EHOOKS trigger block.

To configure the ECU Trigger Delegate Block it should simply be **double-clicked**. This will launch the EHOOKS configuration interface for the EHOOKS trigger delegation block as shown in figure 10.20.



**WARNING**

*The use of the ECU Trigger Delegate Block means that any timing information provided by EHOOKS to the Simulink model will be incorrect. The ECU Trigger Delegate Block should therefore never be used when time-dependent Simulink blocks are used within the model.*

---

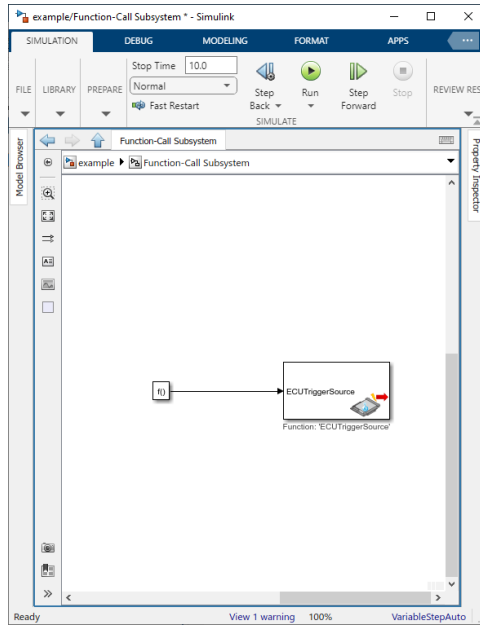


Figure 10.19: EHOOKS ECU Trigger Delegate Block

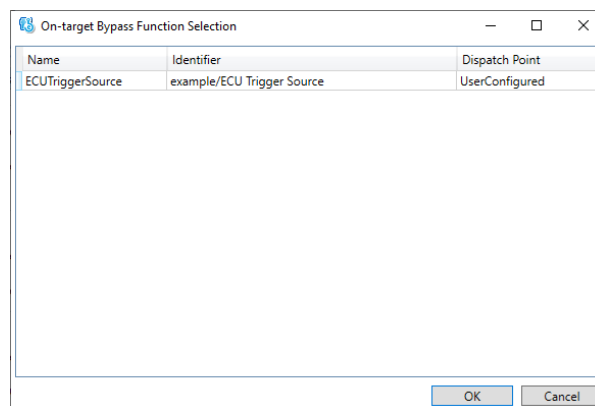


Figure 10.20: EHOOKS ECU Trigger Delegate Block Configuration

### 10.2.7 EHOOKS ECU Hook Control Variable Write Block

The EHOOKS ECU hook control variable write block, shown in figure 10.21, allows a Simulink model to easily write to an EHOOKS hook control variable (see *Control Variables*) to control whether or not an associated hook or on-target bypass function should be enabled.

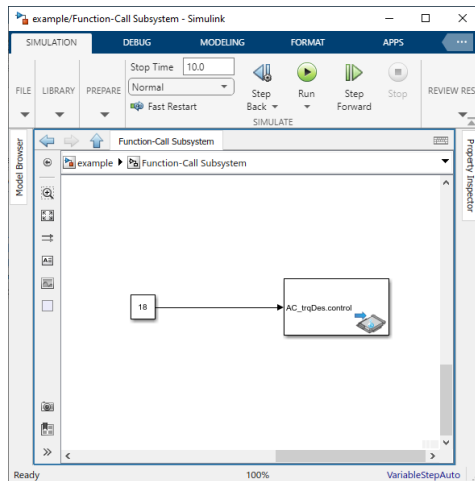


Figure 10.21: EHOOKS ECU Hook Control Variable Write Block

The control variable write block is effectively a data sink block providing a single input port. If a value of 0x12 (18 in decimal) is written to this port, the hooks/on-target bypass functions associated with the configured hook control variable will be enabled. If any other value is written, the hooks/on-target bypass functions associated with the configured hook control variable will be disabled.

To configure the control variable write block, simply **double-click** it. This will launch the EHOOKS configuration interface for the control variable write block, as shown in figure 10.22. Within the text box, the name of the required control variable should be specified to match the name configured within the main EHOOKS-DEV user interface.

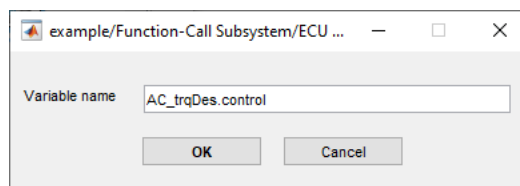


Figure 10.22: EHOOKS ECU Hook Control Variable Write Block Configuration



**NOTICE**

The name configured must match the C name of the EHOOKS hook control variable.

It is important to remember that EHOOKS will automatically convert the name into a valid C identifier if a configured hook control variable name isn't a valid C identifier. This is done by converting any characters that aren't allowed in a C identifier into double underscores (\_\_). For example, a variable name of `AccPed_stNSetP.control` will be converted to `AccPed_stNSetP__control`, and `cyl_pres[2]` will be converted to `cyl_pres__2__`.

The converted C name will need to be used when configuring the control variable write block.

### 10.2.8 EHOOKS ECU Value Parameter Read Block

The EHOOKS ECU Value Parameter Read Block, shown in figure 10.23 allows a Simulink model to read from a scalar calibration parameter that exists within the original ECU software.

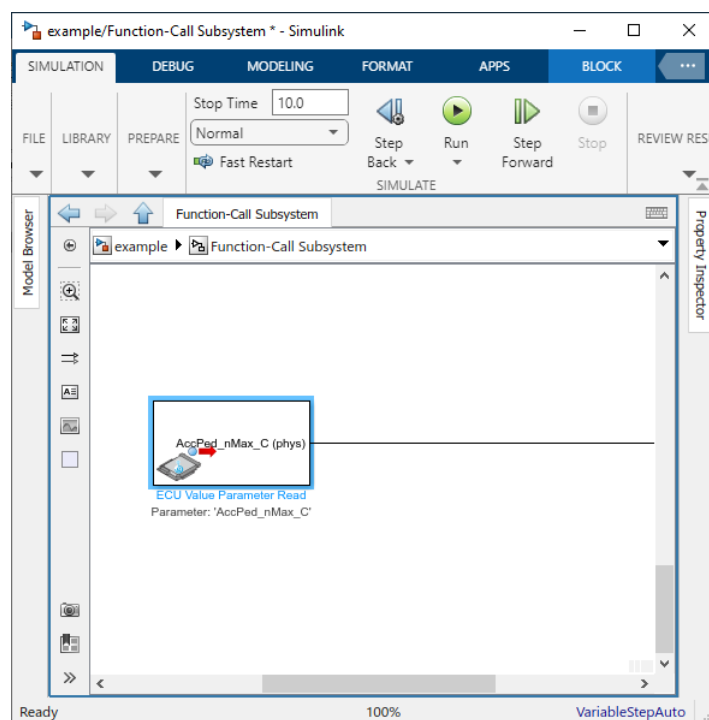


Figure 10.23: EHOOKS ECU Value Parameter Read Block

The ECU Value Parameter Read Block is effectively a data source block providing a single output port. The output port will provide the Simulink model with the current run-time value of a configured ECU scalar calibration parameter.

To configure the ECU Value Parameter Read Block, simply **double-click** it. This will launch the EHOOKS ECU characteristic selection window shown in

Figure 10.24. This window lists all of the available scalar ECU characteristics that can be selected for use in the Simulink model. The name of the ECU characteristic currently used by the block is automatically displayed as the default selection. The list of ECU characteristics

displayed in the window can be filtered by typing a string within the **Filter String** text field. The filter string can include the wildcard characters ? and \*. ? will match any single character and \* will match any number of characters. Note that the filter string contains an implicit \* wildcard at the end of the string. The character \$ can be used to match the end of a characteristic name. As the filter string is updated, the characteristic list will dynamically update to show any matching ECU characteristics that can be read.

The **convert** check-box indicates whether the ECU characteristic value should be converted into physical representation before being provided to the Simulink model or if it should be left in ECU implementation representation. If the check-box is ticked, the associated ECU scalar characteristic value will be converted by EHOOKS to physical representation.

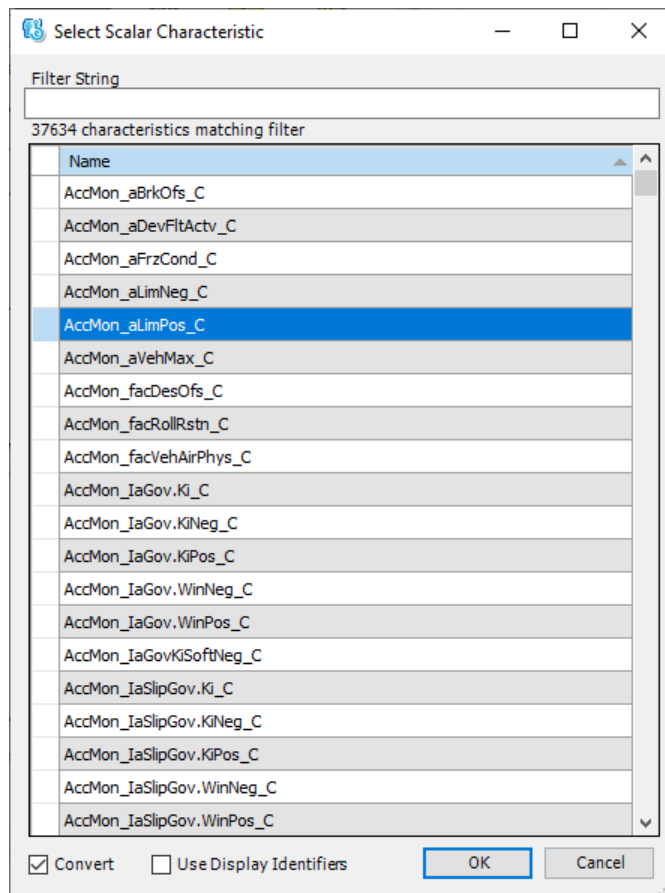


Figure 10.24: EHOOKS ECU Value Parameter Read Block Configuration

Once the **OK** button is selected, the ECU parameter read block will be updated to contain an output port and the name will be updated to match the selected ECU scalar characteristic.

### 10.2.9 EHOOKS ECU Variable Read/Write Blocks

EHOOKS provides three blocks that have a **single** (fixed) read or write port. These blocks are useful when the ECU variables to be read from or written to for particular Simulink signals are changed frequently. The available blocks are:

- ECU Variable Read block
- ECU Variable Write block

- ECU Backup Variable Read block

The configuration process for these blocks is identical to that described in sections 10.2.3, 10.2.4 and 10.2.5 for the ECU variable multiple read/write blocks.

The advantage of using one of the ECU single variable read or write blocks is that the single port is static and so is never disconnected for these new blocks. This attribute is useful for providing a robust mechanism when automated, script- based configuration of statically provided EHOOKS blocks is required.

#### 10.2.10 EHOOKS ECU Function Call Block

When using EHOOKS to perform on-target prototyping experiments, it is often useful to be able to call a function that exists in the original ECU software from a Simulink model. This can be easily achieved by using the EHOOKS ECU Function Call block as shown in figure 10.18.

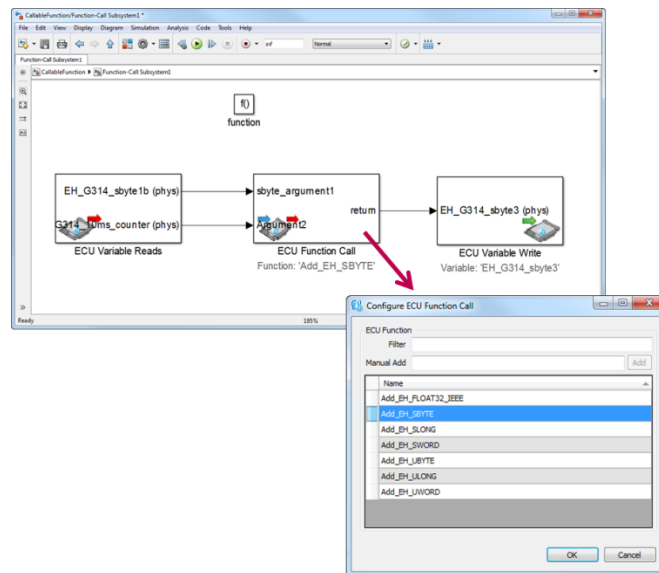


Figure 10.25: EHOOKS ECU Function Call Block

Double clicking on the EHOOKS ECU Function Call block brings up a configuration dialog that includes a list of all of the available ECU functions that it is possible to call from the Simulink model. Select an ECU function to be called from the model by clicking on the name of the function in the list and then click the **OK** button. The name of the ECU function and the correct number of input and output ports are then automatically added to the ECU Function Call block allowing inputs and outputs to be connected.

---

#### NOTICE

*The list of available ECU functions that can be called from a Simulink model is defined by the ECU software supplier as part of the EHOOKS preparation process.*

---

#### NOTICE

*The EHOOKS ECU Function Call block provides support for passing scalar arguments to the ECU function being called. Passing of pointers, arrays and structures is NOT supported. Additionally the use of floating point arguments is not supported for some Renesas SH2 based ECUs.*

---

### 10.2.11 EHOOKS RP-Visible Measurement block

When performing prototyping experiments, it is sometimes useful to be able to feed external signals into a model running on the ECU. To support this use case, it is necessary to write to a measurement inside the model running on the ECU from some code running on external Rapid Prototyping (RP) hardware.

To do this in EHOOKS, an RP-Visible measurement must be created in the Simulink model using the EHOOKS RP-Visible Measurement block as shown in figure 10.26.

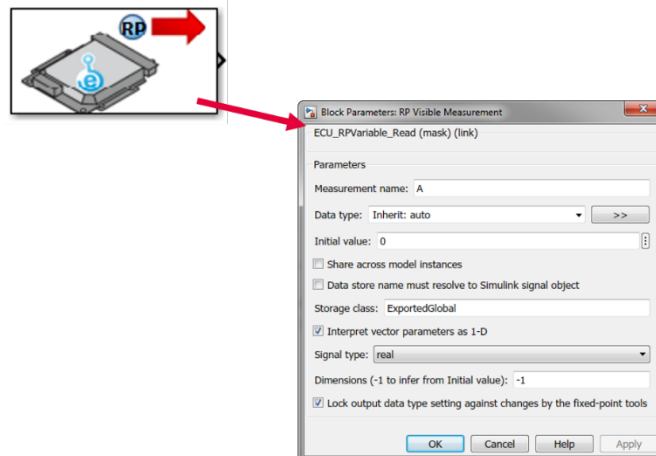


Figure 10.26: EHOOKS RP-Visible Measurement block

The EHOOKS RP-Visible Measurement block is added to the Simulink model. Double clicking on the block will then bring up the dialog shown in figure 10.26, allowing the measurement name, data type and other properties to be configured. The EHOOKS RP-Visible Measurement block is derived from the Simulink 'Data Store Memory' block and hence the configuration parameters and behaviour are the same for both blocks.

When a Simulink model that includes an RP-Visible Measurement is built, EHOOKS will create additional information in the A2L file that allows a Rapid Prototyping tool such as INTECRIO to see and configure a write to the measurement from code running on external RP hardware.

### 10.2.12 EHOOKS ECU Complex Calibration Parameter Read block

The EHOOKS ECU Complex Calibration Parameter Read Block, shown in figure 10.27 allows a Simulink model to read from complex calibration parameters (Maps, Curves, Cuboids) that exist within the original ECU software.

The ECU Complex Calibration Parameter Read Block is effectively a data source block providing a single output port. The output port will provide the Simulink model with the current run-time value of a configured ECU complex calibration parameter.

To configure the ECU Complex Calibration Parameter Read Block, simply **double-click** it. This will launch the Block Parameters dialog shown in figure 10.27. This Block Parameters dialog allows the complex characteristic to be selected, and relevant Simulink options to be configured. By clicking **Select Complex Characteristics**, the dialog shown in figure 10.28 will be displayed.

The EHOOKS Select Complex Characteristics dialog lists all of the available complex ECU characteristics that can be selected for use in the Simulink model. The name of the ECU characteristic currently used by the block is automatically displayed as the default selection.

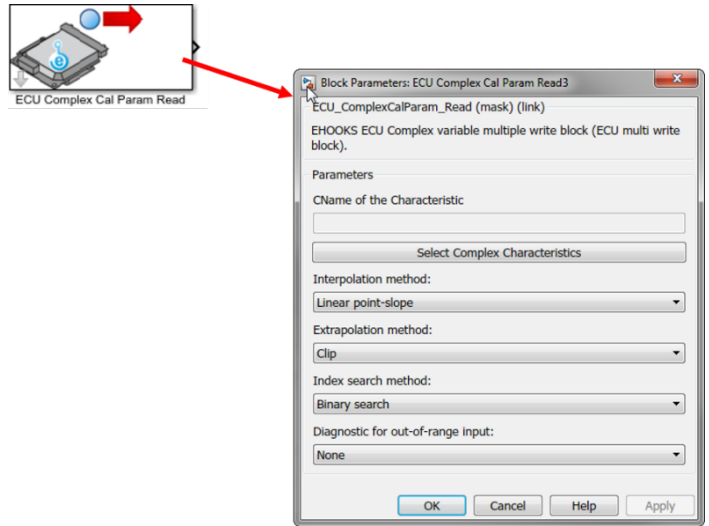


Figure 10.27: EHOOKS ECU Complex Calibration Parameter Read block

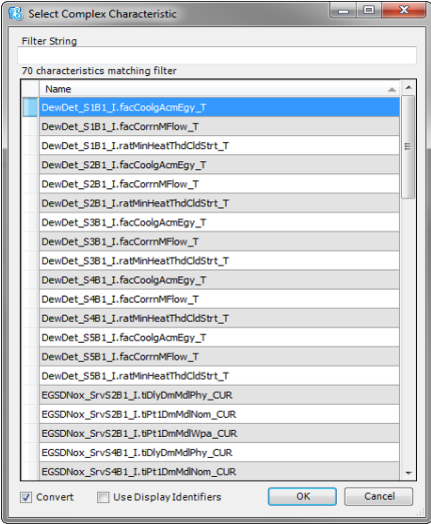


Figure 10.28: EHOOKS Select Complex Characteristics dialog

The list of ECU characteristics displayed in the window can be filtered by typing a string within the **Filter String** text field. The filter string can include the wildcard characters ? and \*. ? will match any single character and \* will match any number of characters. Note that the filter string contains an implicit \* wildcard at the end of the string. The character \$ can be used to match the end of a characteristic name. As the filter string is updated, the characteristic list will dynamically update to show any matching ECU characteristics that can be read.

The **convert** check-box indicates whether the ECU characteristic value should be converted into physical representation before being provided to the Simulink model or if it should be left in ECU implementation representation. If the check-box is ticked, the associated ECU scalar characteristic value will be converted by EHOOKS to physical representation.

The number of inputs to the ECU Complex Calibration Parameter Read block will depend on the type of complex characteristics selected. The number of inputs to the block will be 1 for a Curve, 2 for a Map and 3 for a Cuboid.

The Block Parameters dialog shown in figure 10.27 also allows the configuration of the Interpolation method, Extrapolation Method, Index search method and Diagnostic for out-of-range input. These are Simulink properties of a lookup table which are used when reusing the complex characteristic data.

---

*NOTICE*

Some limitations apply to the use of the ECU Complex Calibration Parameter Read Block.

---

#### 10.2.12.1 Limitations

The limitations to the use of the ECU Complex Calibration Parameter Read Block are as follows.

- The ECU Complex Calibration Parameter Read Block is only supported with versions of Matlab from 2016b onwards
- Reuse of Complex ECU Characteristics is only supported when both of the below parameters are present in the Record layout of a given Characteristic in the a2I file.

```
AXIS_PTS_X / _Y / _Z / _4 / _5
FNC_VALUES
```

- Reuse of Complex ECU Characteristics is only supported when one of the following parameters are present in the Record Layout of the complex characteristics in the a2I file.

```
STATIC_ADDRESS_OFFSETS
STATIC_RECORD_LAYOUT
FNC_VALUES as [ROW_DIR|COLUMN_DIR]
FIX_NO_AXIS_PTS_X / _Y / _Z / _4 / _5
```

- Reuse of MAP and CUBOID characteristics is only supported when the FNC\_VALUES are ROW\_DIR.
- Reuse of CURVE characteristics is supported when the FNC\_VALUES are either ROW\_DIR and COLUMN\_DIR.
- Reuse of Complex ECU Characteristics is not supported when one or more of the below parameters are available in the Record layout of the complex characteristics in the a2I file.

```

    AXIS_RESCALE_X
    DIST_OP_X / _Y / _Z / _4 / _5
    NO_RESCALE_X
    OFFSET_X / _Y / _Z / _4 / _5
    RESERVED
    RIP_ADDR_W
    RIP_ADDR_X / _Y / _Z / _4 / _5
    SHIFT_OP_X / _Y / _Z / _4 / _5
    SRC_ADDR_X / _Y / _Z / _4 / _5
    STATIC_RECORD_LAYOUT without FNC_VALUES as COLUMN_DIR
  
```

### 10.2.13 EHOOKS ECU Value Block Cal Param Read block

The EHOOKS ECU Value Block Cal Param Read block, shown in figure 10.29 allows a Simulink model to read from complex calibration parameters of type VAL\_BLK that exist within the original ECU software.

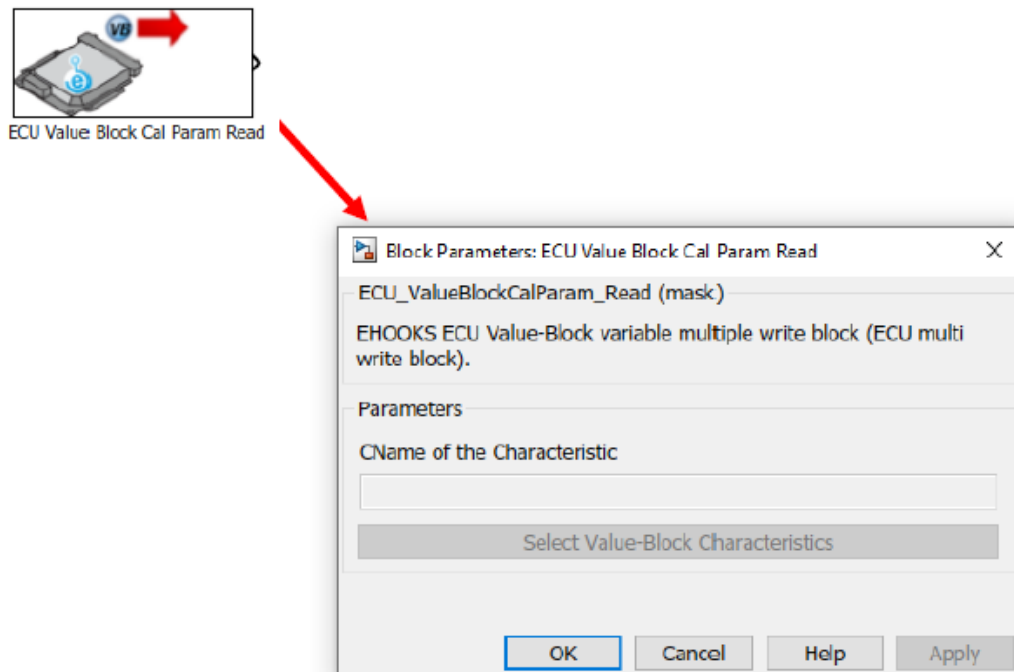


Figure 10.29: EHOOKS ECU Value Block Cal Param Read block

The ECU Value Block Cal Param Read Block is effectively a data source block providing a single output port. The output port will provide the Simulink model with the current run-time value of a configured ECU complex calibration parameter.

To configure the ECU Value Block Cal Param Read Block, simply **double-click** it. This will launch the Block Parameters dialog shown in figure 10.29. This Block Parameters dialog allows the Value-Block Characteristic to be selected. By clicking **Select Complex Characteristics**, the dialog shown in figure 10.28 will be displayed, here you can select the Value-Block Characteristic you wish to re-use.

**NOTICE**

Some limitations apply to the use of the ECU Value Block Cal Param Read block. These are the same limitations as are applied to the reuse of MAP characteristics. Refer to EHOOKS ECU Complex Calibration Parameter Read block for details.

## 10.3 Simulink Modelling for On-Target Bypass

This section describes how to use the EHOOKS blocks to integrate a Simulink model into the ECU software for on-target bypass and build the new ECU software.

### 10.3.1 Adding the EHOOKS Configuration Block

The first step is to create a new Simulink model for our on-target bypass configuration. Next, an EHOOKS configuration block should be added to the top level of the new Simulink model, as shown in figure 10.30.

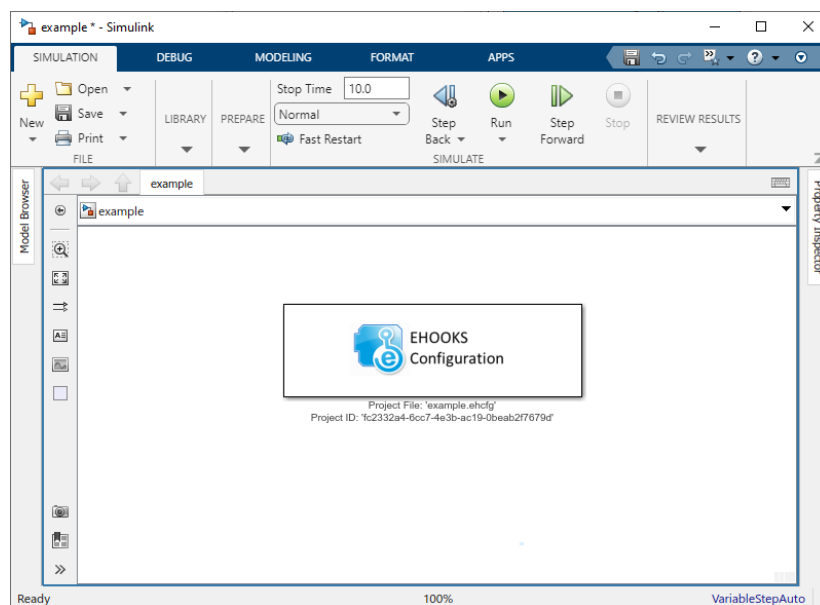


Figure 10.30: A New Simulink Model with an EHOOKS Configuration Block

Next, the EHOOKS configuration block should be **double-clicked** to launch the EHOOKS-DEV configuration interface. Initially the Simulink interface will be displayed to highlight which EHOOKS configuration is associated with the model. At this point a new model can be created and assigned to the model, or an existing EHOOKS configuration can be opened and assigned to the model. After clicking the **Open assigned project...** or **Create and assign project...**, the EHOOKS configuration interface will be launched.

Before continuing with any further modelling or configuration tasks, it is important that at least the input A2L file is configured within the **General Settings** tab, but at this stage it is usually a good idea to complete the necessary configuration of the input HEX file and output HEX/A2L file, and to configure the other general project settings to fit the needs of the experiment (see figure 10.31).

To speed up the subsequent configuration steps it is recommended to simply switch back to the Simulink modeling window and not to close the EHOOKS configuration interface (alternatively the EHOOKS configuration interface could be minimized).



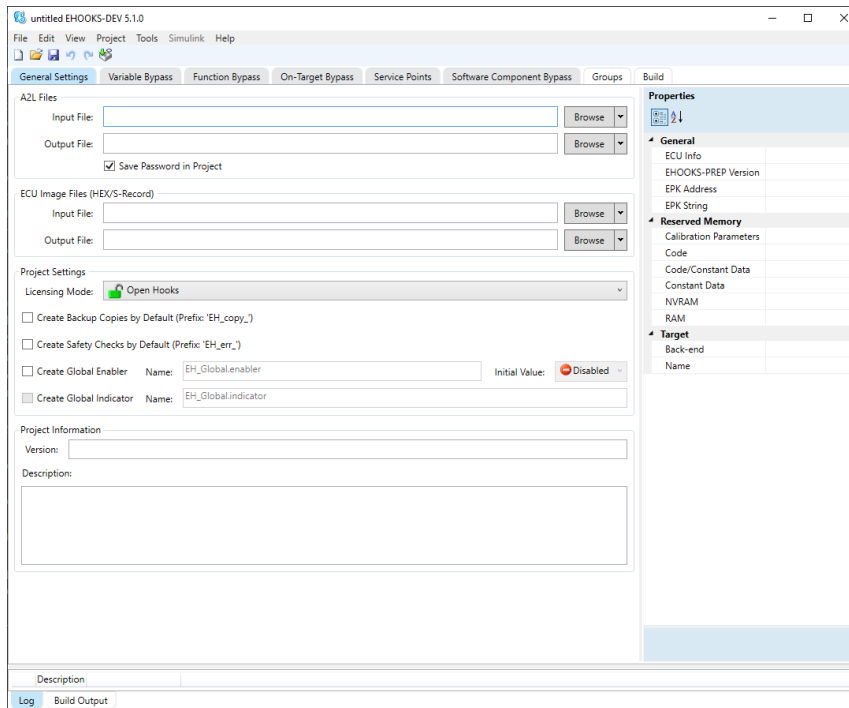


Figure 10.31: Configuring the General Settings

### 10.3.2 Adding the EHOOKS Trigger Blocks

The next step is to consider how the Simulink model for the on-target bypass experiment will be executed within the existing schedule of the ECU software. For simple on-target bypass experiments, it may be sufficient to run the entire Simulink model at once each time a specified ECU process (known as a bypass container) is executed. For more complex experiments, however, it may be necessary to have different model parts execute at different rates within the ECU – perhaps to avoid overloading the processing power of the ECU (by running some complex parts of the model within a slower raster) or perhaps to obtain the desired behavior (by running parts of the model within the raster necessary to achieve the desired functional behavior). Within this chapter, the example will make use of two different rates to execute different parts of the Simulink model to give an idea how this can be achieved. It is, of course, perfectly possible to have significantly more complex experiments where many more rates are used.

For each model part that needs to be integrated into the existing schedule of the ECU software, an EHOOKS ECU trigger block is needed. As the name suggests, an EHOOKS ECU trigger block will trigger the execution of parts of the Simulink model each and every time a specified ECU process is executed. In effect, each EHOOKS ECU trigger block represents a single EHOOKS on-target bypass function. Each EHOOKS ECU trigger block should be connected to a Simulink function-called subsystem (this can be found within the **Simulink library browser** in the **Ports & Subsystems** folder). The function-called subsystem is used as the container for the Simulink model parts that shall be executed when the ECU process associated with the EHOOKS trigger block is executed.

In figure 10.32, the two EHOOKS ECU trigger blocks can be seen. One is configured to run within the ECU process Byp\_100ms\_Proc and another to run within the ECU process Byp\_1000ms\_Proc.

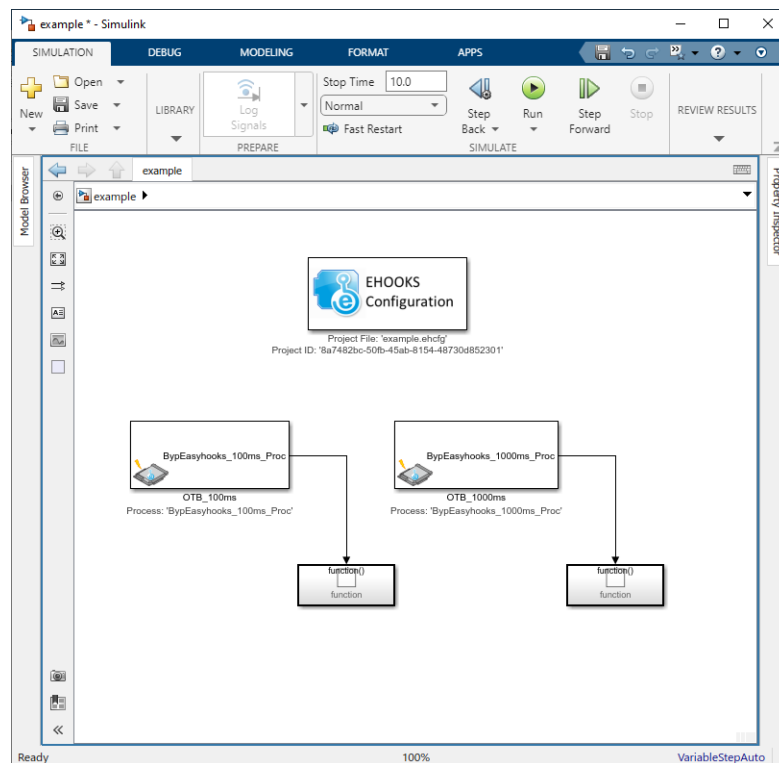


Figure 10.32: Adding ECU Trigger Source Blocks to the Model

---

**NOTICE**

As can also be seen in figure 10.32, the names of the EHOOKS ECU trigger blocks have been changed from the defaults to give more meaningful names. This can be helpful as the name of an EHOOKS ECU trigger block will be used by EHOOKS as the name for the on-target bypass function.

---

Figure 10.33 shows the configuration interface for the EHOOKS ECU trigger block, BypEH00KS\_10ms\_Proc. As can be seen, enablers and indicators have also been configured, so that the execution of these on-target bypass functions can be controlled at run-time via calibration.

### 10.3.3 Adding the Model and Reading/Writing ECU Variables

Typically, the Simulink model parts within the function-called subsystems will now need to be connected into the ECU software by allowing the model to read from and write to ECU data. To do this, EHOOKS ECU variable read and EHOOKS ECU variable write blocks are used.

EHOOKS allows the Simulink model to read and write ECU data either directly in ECU representation or after automatic conversion to physical representation. In the ECU representation case, the port (signal) data type will be the same as the ECU variable data type as specified in the A2L file (if the global option to use floating-point data types described in section 10.2.1 EHOOKS Configuration Block is not set). In the physical representation case, the EHOOKS read and write blocks use the floating-point port (signal) data type.

EHOOKS ECU variable read and write blocks can only be used within a function-called subsystem that is attached to an EHOOKS trigger block. However, as many EHOOKS read and

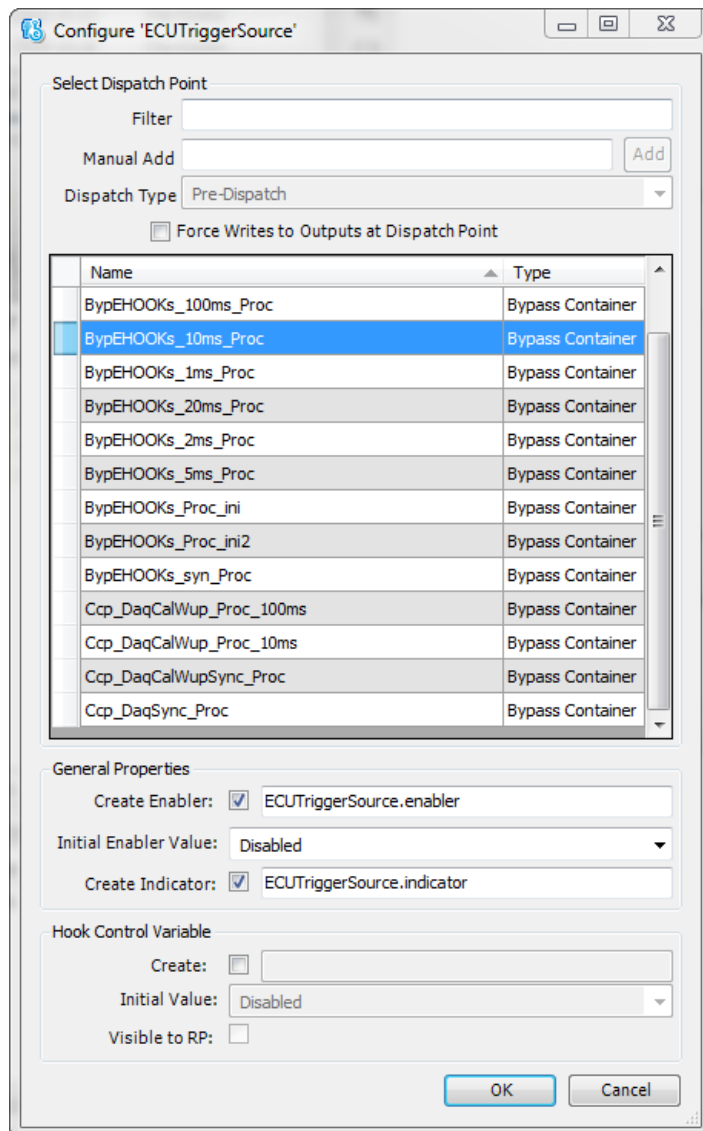


Figure 10.33: Configuring the EHOOKS ECU Trigger Blocks

write blocks can be used within the model as necessary and at any level in the model hierarchy below the function-called subsystem.

In figure 10.34, two EHOOKS ECU variable read blocks and two EHOOKS ECU variable write blocks can be seen. The ports on the blocks indicate the ECU variables being read and written. The annotations on the ports indicate whether the data is being converted between logical or physical representations.

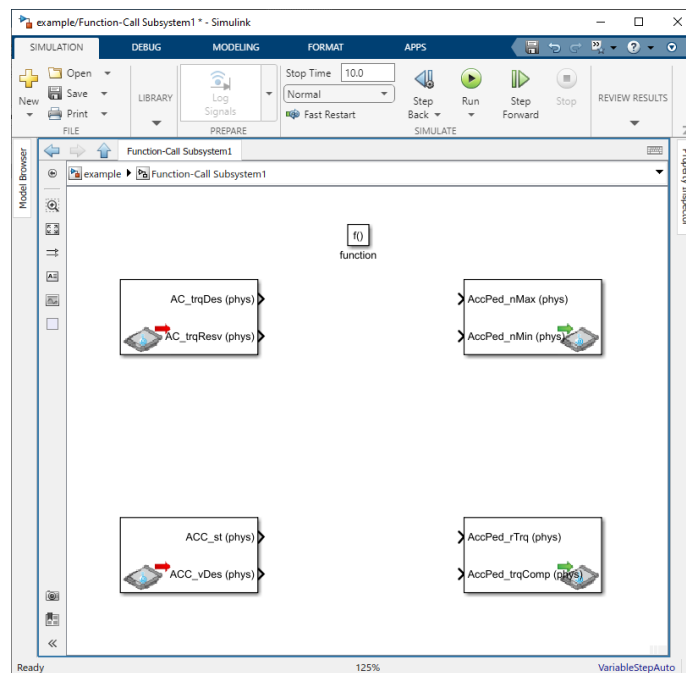


Figure 10.34: Adding EHOOKS ECU Variable Read and Write Blocks to the Model

Figure 10.35 shows the configuration interface for setting the EHOOKS properties associated with the hooks that EHOOKS will create for each variable contained in an EHOOKS ECU variable write block. For example, the variable EH\_G314\_sbyte3 has been configured to include a backup copy measurement.

#### 10.3.4 Adding the Simulink model

Once the EHOOKS ECU variable read and write blocks are added to the model, all that remains is to connect them with the functional Simulink model to perform the desired new control strategy. Figure 10.36 shows an example with a very simple model (the model shown is just a dummy model to illustrate the usage – typically the functional Simulink model may be much more complex and include the use of additional subsystems).

##### NOTICE

The direct connection of an EHOOKS ECU variable read block to a Simulink merge block is not supported by default. A Simulink merge block compatible block should be placed between the ECU Variable Read block and the Merge block.

If storage reuse is required directly for the EHOOKS ECU variable read block then it must be enabled via the EHOOKS block automation interface as described in section 11.3.2 EHOOKS Simulink APIs. Enabling storage reuse is at your own risk because it may lead to data corruption due to over-aggressive signal variable reuse by Simulink.

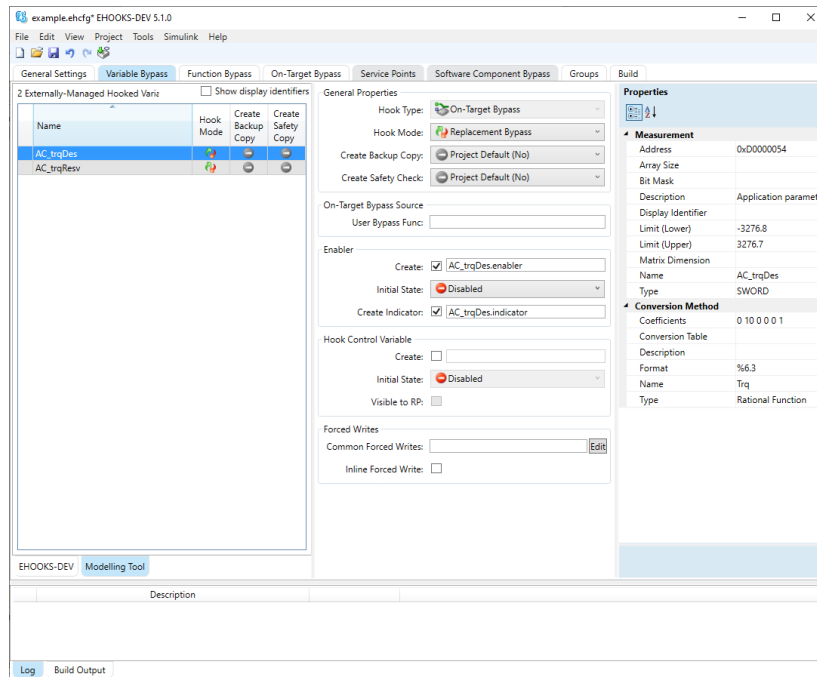


Figure 10.35: Configuring the Hook Properties for ECU Variable Writes

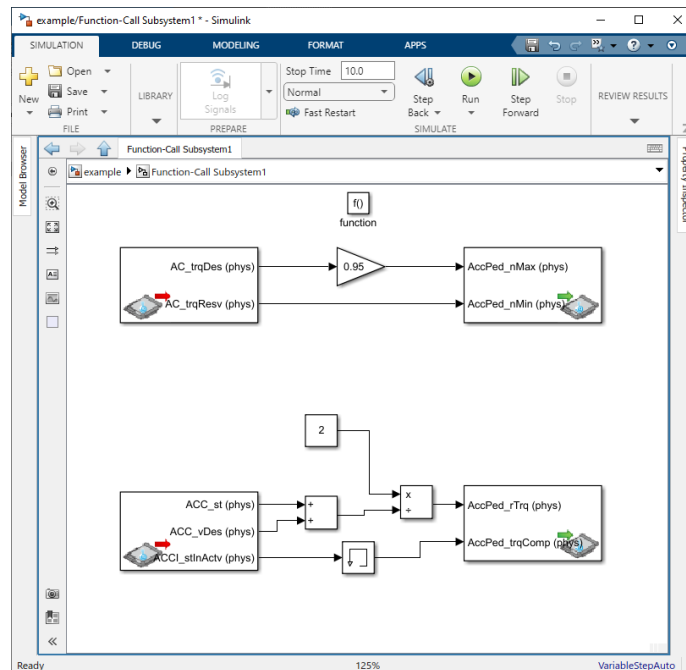


Figure 10.36: Adding the Functional Simulink Model

## 10.4 Building Hooked ECU Software with Simulink On-Target Bypass

This section describes how to configure Simulink to allow the EHOOKS on-target bypass model set-up to be built, how to perform the build of the newly hooked ECU software and how to perform an experiment on the ECU with the newly hooked ECU software using INCA.

### 10.4.1 Setting the Simulink Configuration Properties

To be able to build the on-target bypass model, it is necessary to first set some configuration options. To display the configuration options, the Simulink model explorer should be launched from the **View -> Model Explorer** (Ctrl+H) menu within Simulink. Then the active configuration should be modified as follows and as shown in figure 10.37.

Configuring the Simulink Model for use with EHOOKS

- Configuration Solver

Within the **Solver options** section, the **Type** should be set to **Fixed-Step** and the **Solver** set to **discrete (no continuous states)**. EHOOKS only supports the discrete solver in the current version. If the solver type is not manually set to **Fixed-step/discrete**, then EHOOKS will automatically configure it to be **Fixed-step/discrete**.

- Configuration Optimization

Within the **Simulation and code generation** section, it is recommended (but not absolutely necessary) to tick the **Inline parameters** check-box. This can significantly reduce the RAM requirements of the generated code and, as the model will need to execute within the constrained environment of the ECU, this is a good idea.

- Configuration Real-Time Workshop General Tab

Within the **Target selection** section, the **System target file** should be changed to one of:

- ehooks\_grt.tlc / EHOOKS Real-Time Target

Selecting this target will use the Simulink Real-Time Workshop code generator to create the necessary C code from the Simulink model.

- ehooks\_ert.tlc / EHOOKS Real-Time Target for Embedded Coder

Selecting this target will use the Simulink Real-Time Workshop Embedded Coder code generator to create the necessary C code from the Simulink model. This code can be significantly more efficient and more suitable for running within the constrained environment of the ECU. However, a valid MATLAB / Simulink Real-Time Workshop Embedded Coder license is required to use this target.

- Configuration Real-Time Workshop Interface Tab

Within the **Software environment**, the **Utility function generation** must be changed to **Shared Location**. If the utility function generation is not set to **Shared Location**, then EHOOKS will automatically configure it to be **Shared Location**.

- Configuration Real-Time Workshop EHOOKS Tab

If necessary, any additional command-line options to the EHOOKS ECU Target Support tools can be specified here.

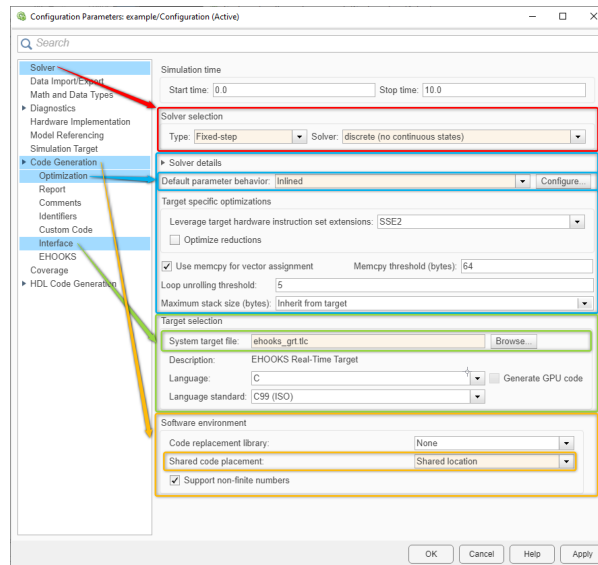


Figure 10.37: Setting the Simulink Configuration Properties

**NOTICE**

Simulink will generate some warnings when building a model with EHOOKS blocks. Generally, these warnings are benign and can be ignored. However, all warnings should first be reviewed to ensure that important build problems aren't indicated. The benign warnings due to the use of the EHOOKS blocks can be disabled by setting the following diagnostic options within the **Diagnostic** page of the **Simulink Configuration Properties** (additional caution should be taken when doing this to ensure valid warnings aren't hidden from view):

- *Tasks with equal priorities: None*
- *Source block specifies -1 sample time: None*

**NOTICE**

When supplying additional command line arguments to EHOOKS, any paths (e.g. for the `--logdir` option) must use either `'/'` (forward slash) or `'\\'` (double backslash) as path separators – e.g. `D:\\logdir` or `D:/logdir`

#### 10.4.2 Building the Hooked ECU Software

Once the configuration options have been set, the standard Simulink build process will fully integrate EHOOKS and create the new hooked ECU software. To begin the build process, simply select the **Tools Real-Time Workshop Build Model** menu item or press **Ctrl+B**. The progress of the build will be logged in the MATLAB command window, including the output from all the EHOOKS build steps. All information, warnings and errors will be reported in this log.

#### 10.4.3 Running an Experiment with the Hooked ECU Software using INCA

The first step in running the experiment is to flash the hooked ECU software to the ECU. This should be carried out in the usual manner (typically using INCA ProF). Once the ECU has been

flashed with the newly created hooked ECU software, an INCA experiment can be started. Figure 10.38 shows an INCA experiment for the example Simulink model created in this section.

To work with the new Simulink functionality added into the hooked ECU software, the configured enablers first have to be activated. In the example setup, it can be seen that we have a global enabler, an enabler for the two on-target bypass functions modelled with Simulink and an enabler for each variable written to by an EHOOKS ECU variable write block in the Simulink model. These can all be managed separately. Of course, if the global enabler is false, all of the software changes introduced by EHOOKS into the hooked ECU software will be inactive. If a variable hook enabler is true but the corresponding on-target bypass function enabler is false, the variables will not be bypassed as the new function (model) will not be executed.

In figure 10.38, it can be seen that the value of the AFS\_dmDrft variable has been bypassed by the Simulink model to contain the value coming from the variable AFS\_dm. The sequence counter which was modeled (but not shown in this section) in the OTP\_1000ms on-target bypass function is executing and is counting the number of executions.

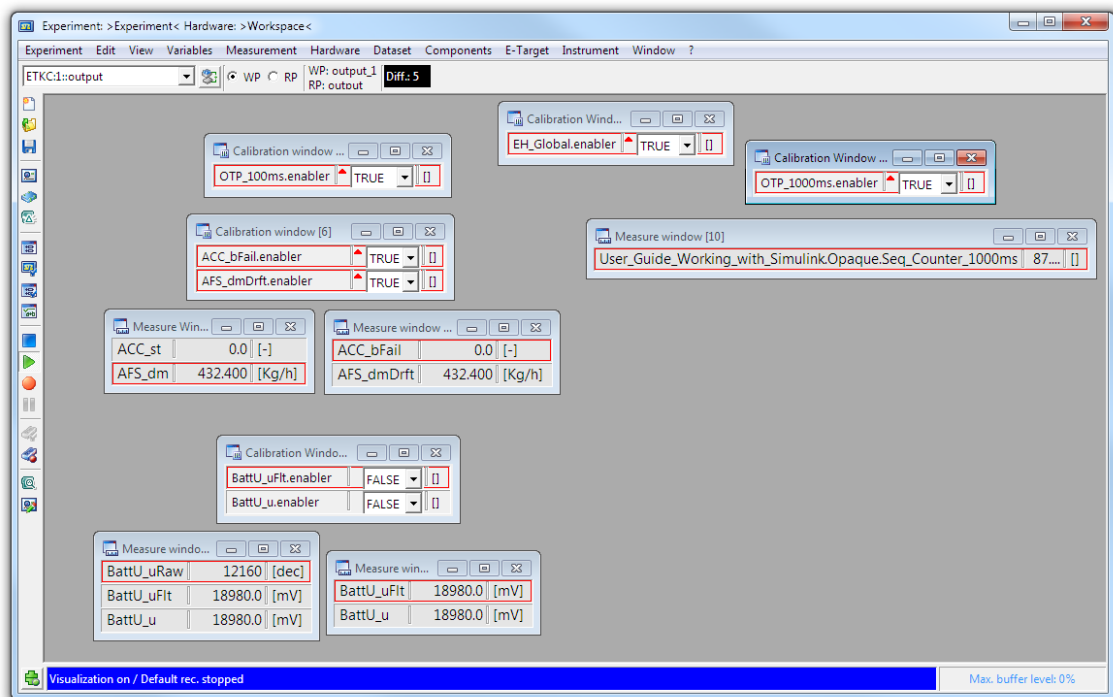


Figure 10.38: INCA Experiment

## 10.5 Advanced Simulink Features

This section details some of the more advanced features that are available when using Simulink for on-target bypass within the EHOOKS-DEV Simulink Integration Package.

### 10.5.1 Creating Model Measurements and Calibration Data

For all but the simplest Simulink models it will be helpful and important to be able to create new measurement and calibration data for the model contents. This allows the new functionality added through a Simulink model to be measured and calibrated in the same way as the existing parts of the ECU software.



### 10.5.1.1 Creating New Measurement Data

To make a Simulink signal measurable in the hooked ECU software is simply a matter of giving the signal a name and setting its Real-Time Workshop storage class to **ImportedExtern**. Figure 10.39 shows how to do this. A signal can be assigned a name in Simulink either by **double-clicking** on the signal line and entering the name or via **Right-Click -> Signal Properties** and completing the **Signal name** text field.

Once a name has been defined the storage class can be changed to ImportedExtern via the **Right-Click -> Signal Properties** dialog on the **Real-Time Workshop** tab.

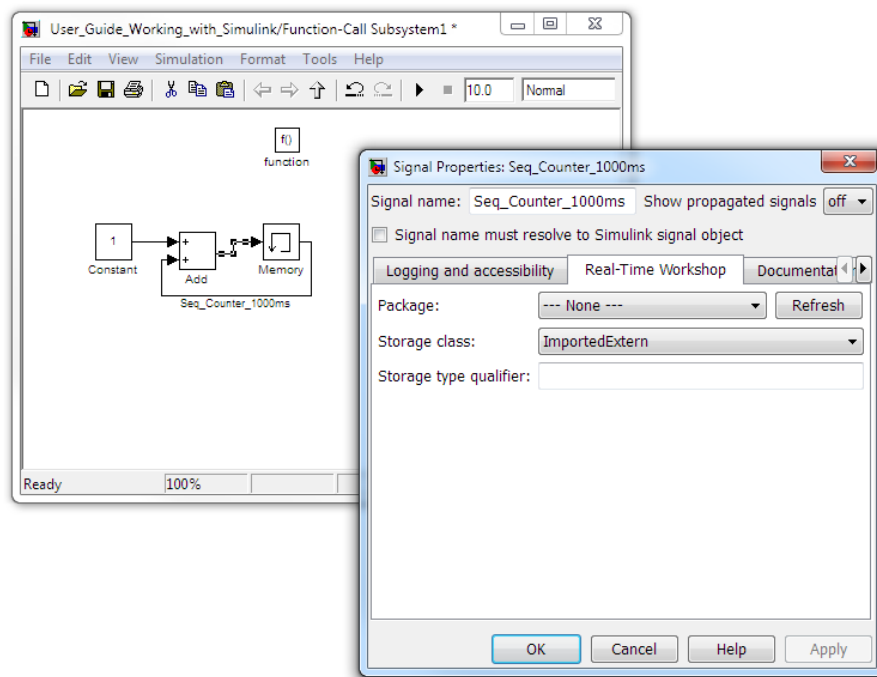


Figure 10.39: Making a Simulink Signal into an ECU Measurement

It is also possible to define Simulink signals in the MATLAB base workspace as described later for Simulink parameters and shown in figure 10.40. In this case, the Simulink signal must resolve to a global Simulink signal object (see according checkbox in figure 10.39). The global Simulink signal object must also be configured to use the Real-Time Workshop storage class **ImportedExtern**.

### 10.5.1.2 Creating New Calibration Data

To make a Simulink parameter available as a calibration parameter in the hooked ECU software, set the storage class of the associated MATLAB variable or Simulink parameter defined in the MATLAB base workspace to **ImportedExtern**. The process to achieve this is a little different depending whether MATLAB variables or Simulink parameters are used. Using Simulink parameters is recommended as they offer the ability to control the data-type used in the implementation, whereas MATLAB variables are always floating point values. However, both MATLAB variables and Simulink parameters are fully supported.

As a first example, a simple scalar in our model will be made into a calibration parameter. In figure 10.36, the gain block used has a fixed static gain of 0.95. To make this gain a calibration parameter that can then be controlled at run-time via INCA, the following steps should be performed:

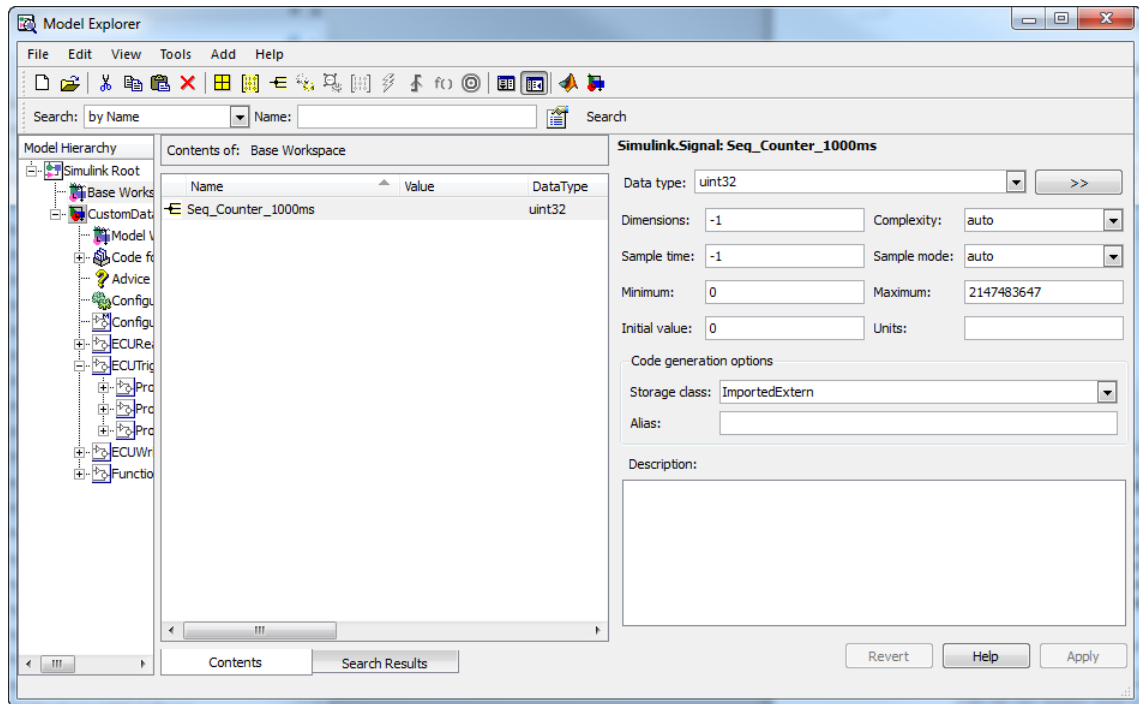


Figure 10.40: Configure a Simulink Signal as a Measurement Variable

### 10.5.1.2.1 Creating a Simple Scalar Calibration Parameter

- Open the parameter dialog for the gain block via **double-click** or **right-click** -> **Gain Parameters....**
- Within the gain parameter, enter the name of the new calibration parameter to be created, as shown in figure 10.41.

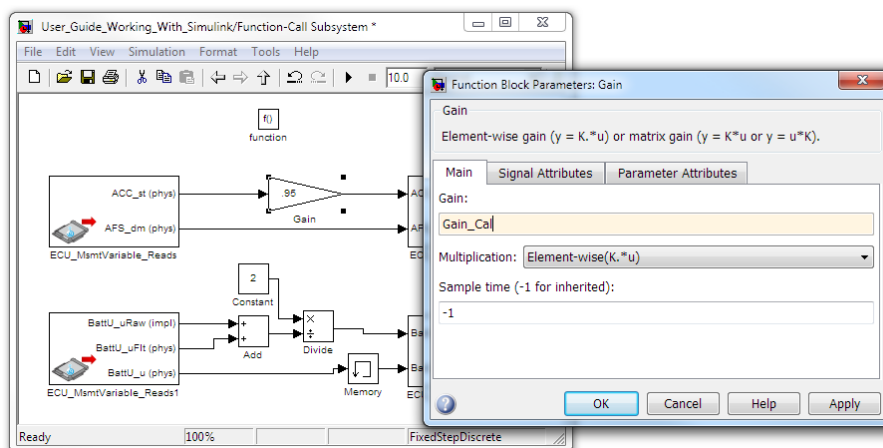


Figure 10.41: Configuring the Gain Block

The subsequent steps vary depending on whether a MATLAB variable or Simulink parameter is used:

### 10.5.1.2.2 Using a MATLAB Variable as a Calibration Parameter

- Within the Simulink Model Explorer (Ctrl+H) Base Workspace, create a new MATLAB variable via the **Add -> MATLAB Variable** (Ctrl+M) menu.
- Set the name of the MATLAB variable to match the calibration parameter name configured above, as shown in figure 10.42.
- Set the initial value for the MATLAB variable. This will be used by EHOOKS as the initial value for the created calibration parameter within the calibration data reference page.
- Within the active configuration select the Optimization setting.
- Ensure that the **Default parameter behaviour** is set to **Inlined** within the **Signals and Parameters** section and click **Configure...**
- Within the **Model Parameter Configuration** dialog that is displayed, select the created MATLAB variable and click Add to table >>.
- Change the Storage class of the MATLAB variable to **ImportedExtern**, as shown in figure 10.43.

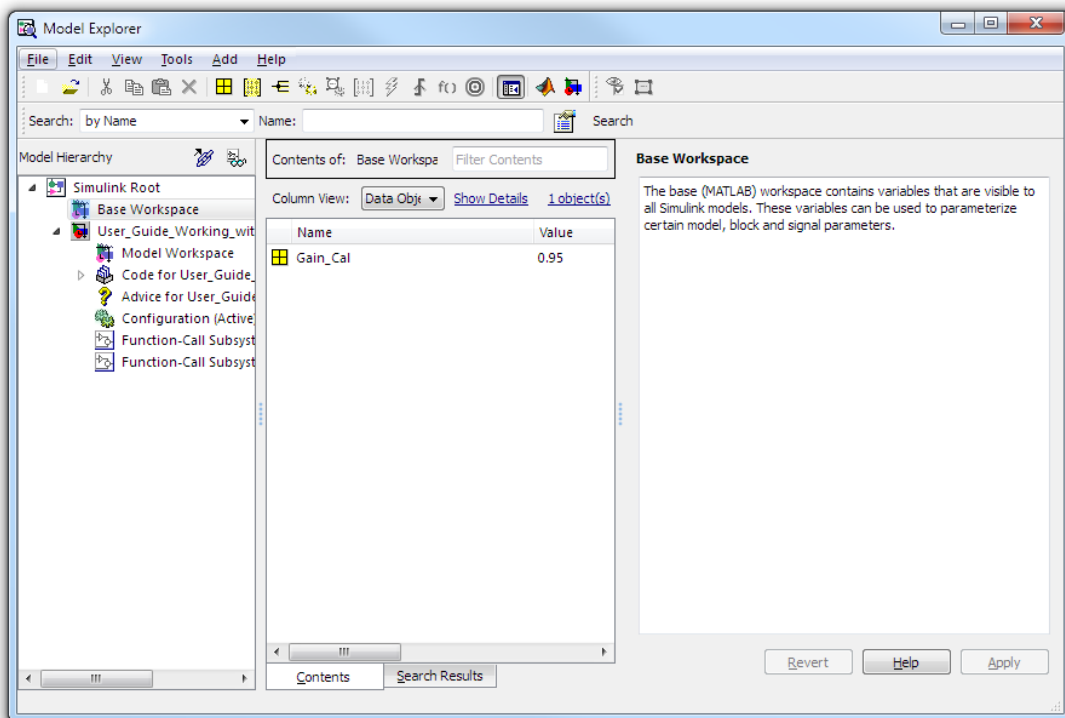


Figure 10.42: Adding the MATLAB Variable

### 10.5.1.2.3 Using a Simulink Parameter as a Calibration Parameter

- Within the Simulink Model Explorer (Ctrl+H) Base Workspace, create a new Simulink parameter via the **Add -> Simulink Parameter** (Ctrl+P) menu.
- Set the name of the Simulink parameter to match the calibration parameter name configured above.
- Set the initial value for the Simulink parameter. This will be used by EHOOKS as the initial value for the created calibration parameter within the calibration data reference page.

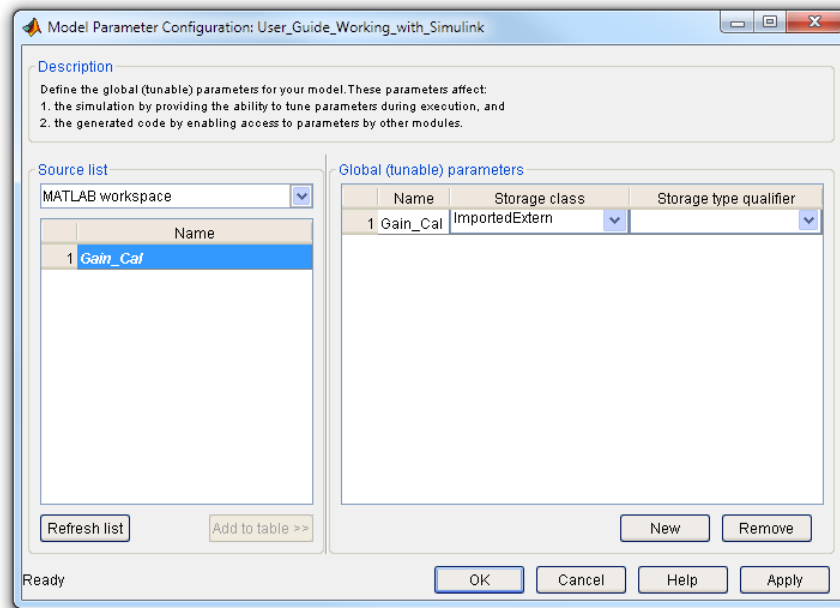


Figure 10.43: Configure a MATLAB Variable as a Calibration Parameter

- Within the active **configuration** select the Optimization setting.
- Ensure that the **Inline parameter** check-box is ticked within the **Simulation and code generation** section.
- Set the data type as required.
- Set the Storage class to **ImportedExtern**, as shown in figure 10.44.

The same basic steps are followed when creating more complex types of calibration parameters such as maps and curves – represented in Simulink with **Lookup Table** Blocks. For example, figure 10.45 shows how to create a map calibration parameter called Batt\_LUT. Provided that MATLAB variables/Simulink parameters used for the row, column and data elements are all marked as having the ImportedExtern storage class, EHOOKS will automatically create a map calibration parameter in the created A2L file for the hooked ECU software.

---

**NOTICE**

*EHOOKS fully supports sharing map/curve axis data between several maps/curves. To do this, simply ensure the same MATLAB variable/Simulink parameter is used for the row/column data of the associated Lookup Table blocks.*

---

### 10.5.1.3 Using Prefixes and Suffixes for Measurement and Calibration Data

The arguments `--a2lname-prefix=<prefix>` and `--a2lname-suffix=<suffix>` can be used to work around name conflicts with elements that are already present in the ECU project (see also section 11.1 EHOOKS-DEV Command Line Usage).

These arguments can be specified at **Configuration Parameters -> Real-Time Workshop -> EHOOKS -> Additional back-end arguments**. Automated setup of the arguments can be done via an M script that sets the EHOOKS specific model configuration set parameter

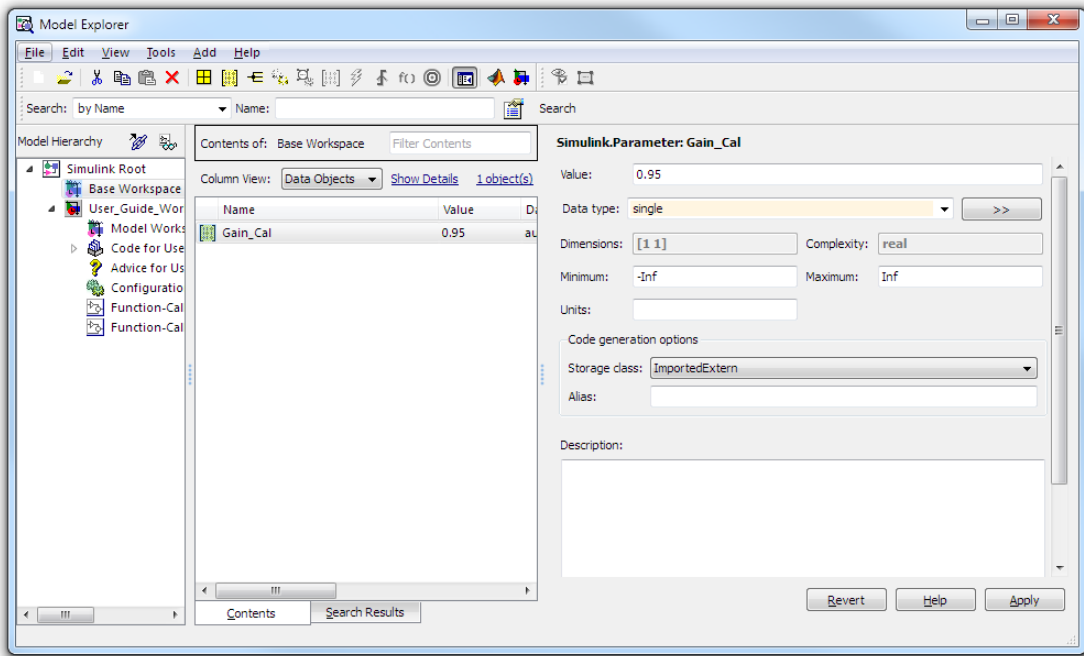


Figure 10.44: Configure a Simulink Parameter as a Calibration Parameter

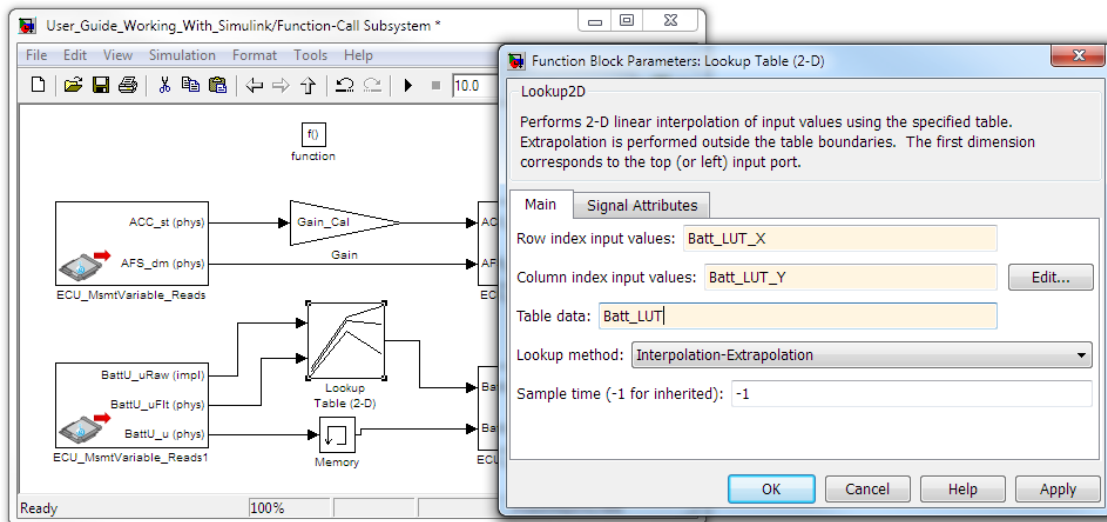


Figure 10.45: Adding a Map Calibration Parameter

EHAdditionalBackEndArguments , e.g.

```
set_param(bdroot, 'EHAdditionalBackEndArguments', ...
    ' --a2lname-prefix=EH_ --a2lname-suffix=_RP').
```

#### NOTICE

The `--use-cids-as-a2lnames` option has been removed due to a change in the nature of A2L element naming from Matlab Simulink integration. Global elements are no longer generated with complex paths, and as such, this option is redundant in such cases. Using this option will no longer have any affect on the generated A2L, and ALL A2L elements will be named according to the C identifier specified in the modelling tool directly.

### 10.5.2 Reading Existing Scalar and Complex ECU Calibration Data

The EHOOKS-DEV Simulink Integration Package provides the EHOOKS ECU parameter read block to allow scalar calibration parameters within the existing ECU software to be reused within the Simulink model. To do this, simply add an ECU parameter read block to the model. Double-click the EHOOKS ECU parameter read block and enter the name of the existing ECU calibration parameter to be read and configure whether it should be converted to physical representation or provided directly in implementation representation. Then connect the output port of the EHOOKS ECU parameter read block into the model, as shown in figure 10.46.

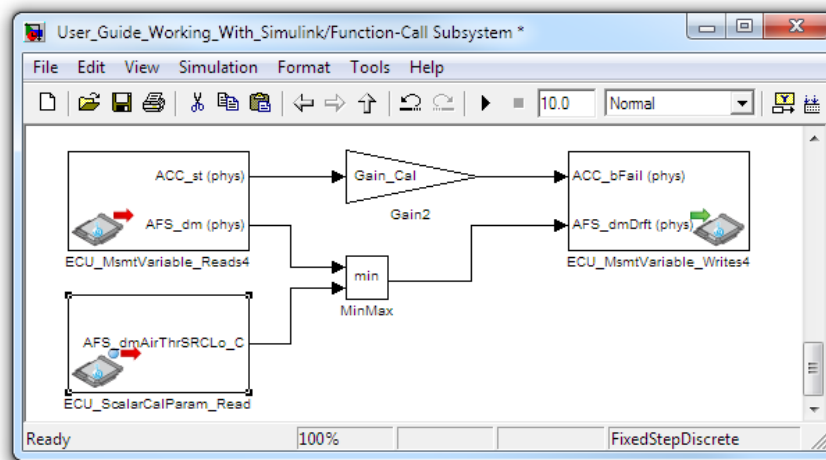


Figure 10.46: Using EHOOKS ECU Value Parameter Read Blocks

The same process can also be used to reuse complex calibration parameters within the existing ECU software by replacing the EHOOKS ECU parameter read block in the example above with the EHOOKS ECU complex calibration parameter read block.

### 10.5.3 Reading from Hooked ECU Variable Backup Copies

It can often be helpful to access the original ECU calculation for a variable that is being hooked and bypassed by EHOOKS. This can be useful if, for example, you want to filter or calculate an offset to the original ECU calculation for the variable. For this, the ECU backup variable read block can be used. First, a variable hooked by EHOOKS (either directly in EHOOKS or via an EHOOKS ECU variable write block) must be configured to **Create Backup Copy**, as shown in figure 10.47.

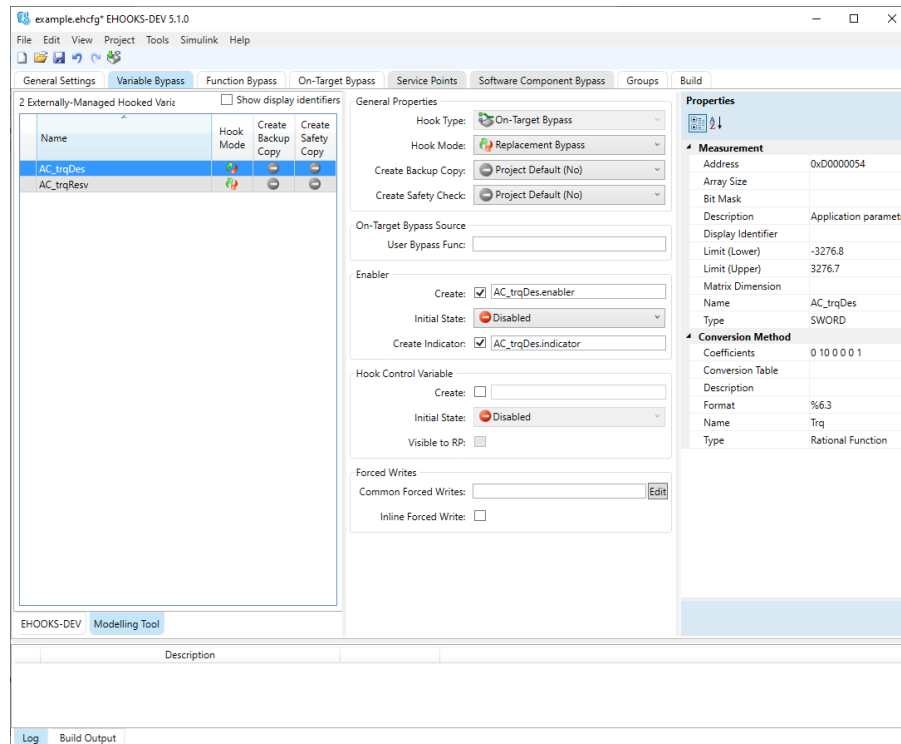


Figure 10.47: Configuring EHOOKS to Create a Backup Copy

Then the EHOOKS ECU backup variable read block can be used to provide access to the original ECU calculation for a variable before the bypass value was applied as shown in figure 10.48.

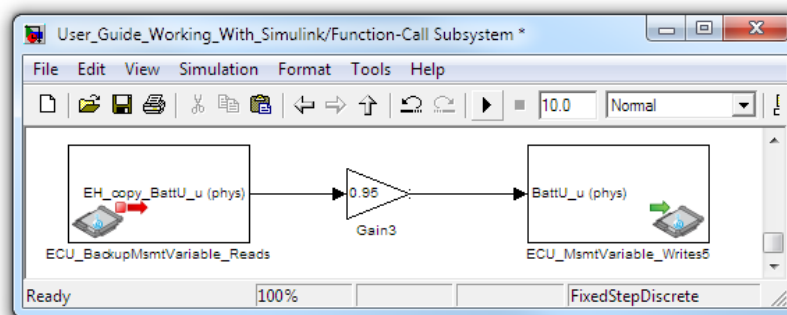


Figure 10.48: Example Use of EHOOKS ECU Backup Copy Variable Read Block

#### 10.5.4 Programmatic Control using Control Variables

Sometimes it can be necessary to control whether specific hooks or on-target bypass functions are enabled, not only by calibration parameters but also programmatically at run-time. EHOOKS provides this ability via control variables that act just like the calibration enablers, apart from the fact that they are managed at run-time by C-code or Simulink model code rather than INCA.

Using control variables is straight-forward. First, the control variable must be defined and associated with a specific variable hook or on-target bypass function. For example, figure

10.49 shows a control variable, `OTP_1000ms_control`, being defined for the on-target bypass function `OTP_1000ms`.

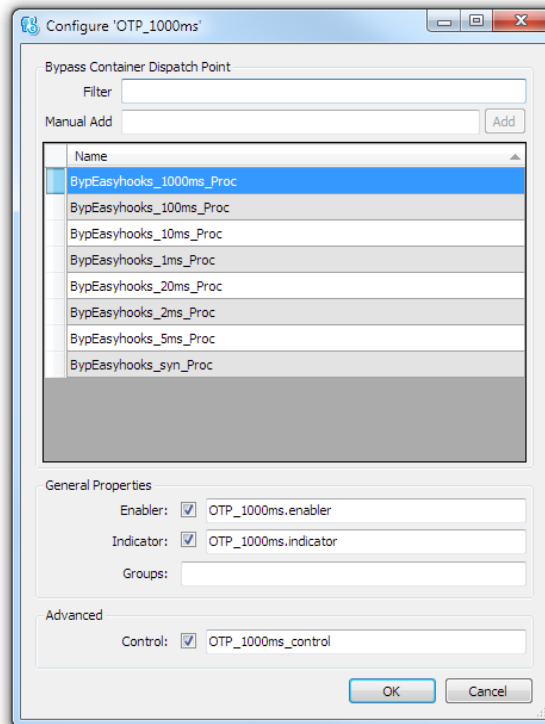


Figure 10.49: Defining a Control Variable for an On-Target Bypass Function

This control variable can then be used by an EHOOKS ECU control variable write block to allow the Simulink model to control whether or not this on-target bypass function is activated. Figure 10.50 shows an example of this, where a Simulink if block is used to check a condition before enabling or disabling the control-variable.

#### 10.5.5 Communication between On-Target Bypass Functions

Most reading and writing of ECU data and model calculations will typically occur within the function-called subsystems which are attached to EHOOKS ECU trigger blocks. It can sometimes be necessary, however, to allow communication between the different on-target bypass functions represented by the EHOOKS ECU trigger blocks.

As each EHOOKS trigger block represents an asynchronous trigger, signals cannot be directly connected between the attached function-called systems. Instead, rate transition blocks should be used. Figure 10.51, shows how these blocks can be used to successfully allow state calculated by one on-target bypass function to be communicated to another.

#### 10.5.6 Trigger Delegation

On-target bypass functions are typically triggered by an ECU process known as a bypass container. It can sometimes be useful, however, to have one on-target bypass function trigger the execution of another on-target bypass function directly. The EHOOKS ECU trigger delegation block allows such direct triggering. To configure trigger delegation, simply add an EHOOK ECU trigger delegation block into the Simulink model and connect it to a standard function-call generator block, as shown in figure 10.52. This will then directly trigger the execution of the configured on-target bypass function when this part of the Simulink model



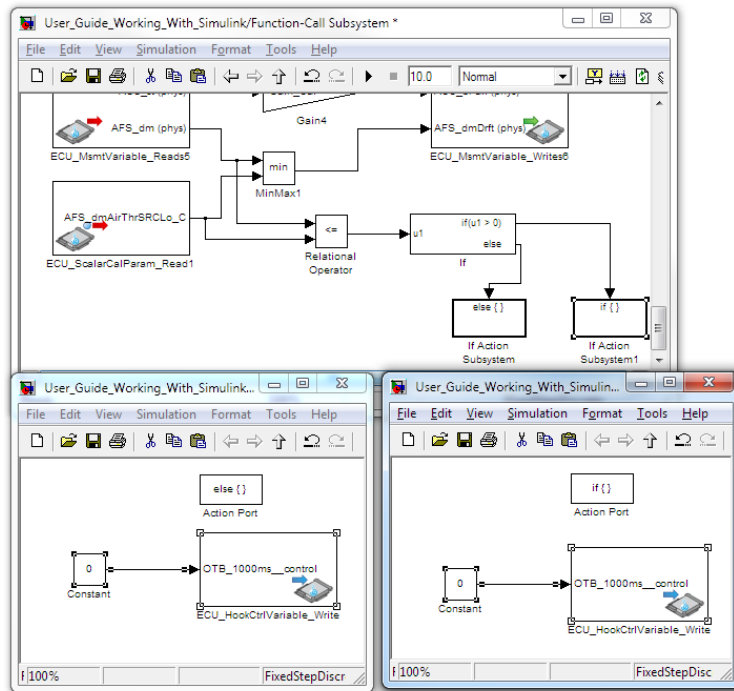


Figure 10.50: Using the EHOOKS ECU Hook Control Variable Write Block

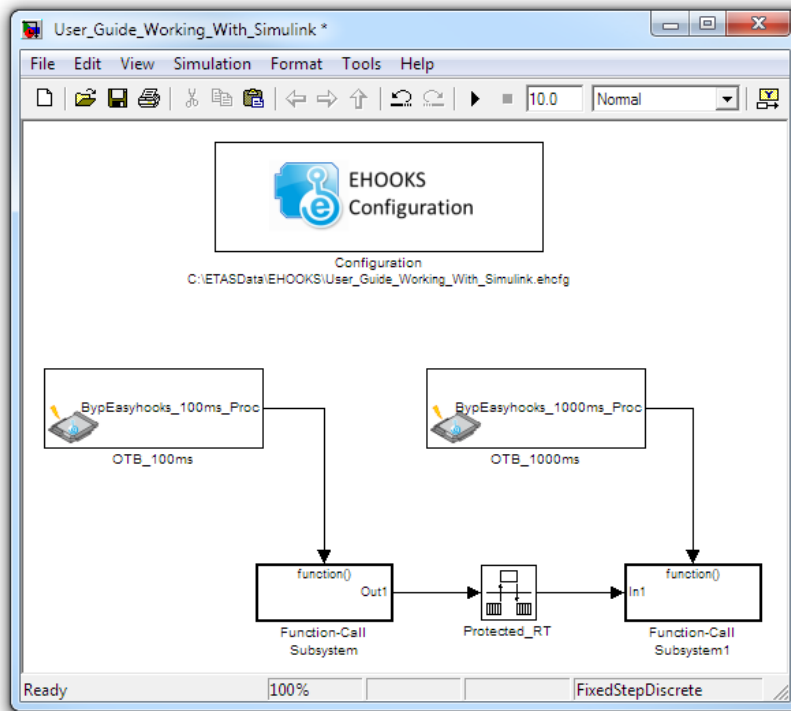


Figure 10.51: Using Rate Transition Blocks

code is executed.

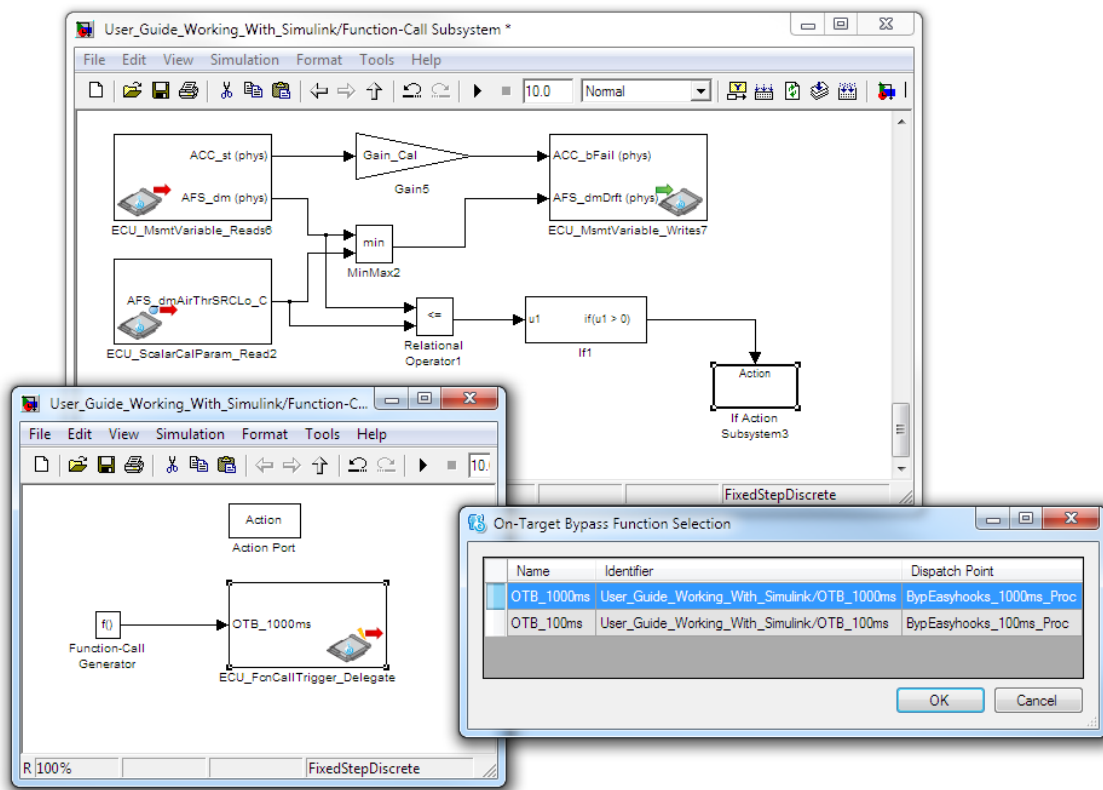


Figure 10.52: Using a EHOOKS ECU Trigger Delegation Block

### 10.5.7 Calling an ECU function from within a Simulink model

It is often useful to be able to call an existing ECU function from within a Simulink model. This is possible by either using an EHOOKS ECU function Call block in the Simulink model (see section 10.2.10 EHOOKS ECU Function Call Block), or by using the Simulink Legacy-Code feature.

To call an existing ECU function called ExampleFunc using the Simulink Legacy-Code feature, the following steps should be taken:

Step 1) Create a header file (for example: myfunction.h) containing the prototype of ExampleFunc and a function pointer to it. This allows the C code generated by Simulink/RTW to call through the function pointer to the function ExampleFunc inside the .hex/s19 file. The file myfunction.h would therefore contain the following:

```
typedef EH_FLOAT32_IEEE (*EH_ExampleFunc_type)(EH_FLOAT32_IEEE, EH_FLOAT32_IEEE);
#define EH_ExampleFunc ((EH_ExampleFunc_type)(0x800B38A0))
```

The above code defines a function pointer type for the function ExampleFunc and then defines the address in the ECU hex/s19 file for this function.

This header file should be updated each time the application code is rebuilt as the address of the function is likely to change during each build.

Step 2) Use the Simulink Legacy Code Tool to generate and compile an S-function block to call this code. The following Matlab command line options can be used to do this:

```
def = legacy_code ('initialize');
def.HeaderFiles = {'myfunction.h'};
```

Tells RTW to #include this header file when this s-function block is used.

```
def.SFunctionName = 'ExampleFunc';
```

defines the name of the generated s-function.

```
def.OutputFcnSpec = 'float y1 = EH_ExampleFunc (float u1, float u2)';
```

provides the function prototype. Simulink data types must be used.

```
legacy_code('sfcn_cmex_generate', def);
legacy_code('compile', def,
            '-IC:\Program Files\MATLAB\R2015a\simulink\include');
```

generates and compiles the S-function block implementation.

```
legacy_code('slblock_generate', def);
```

creates a new model file which contains the s-function block that has been generated to represent the ECU function to be called.

```
legacy_code('sfcn_tlc_generate', def);
```

generates the TLC code to allow code-generation for the new block

Step 3) The generated block from step 2 can then be used in a Simulink model so that when code is generated a function call is made (via the function pointer) into the original ECU code that implements the function ExampleFunc.

---

**NOTICE**

*For further details on the use of the Simulink Legacy Code Tool please refer to the Matlab/Simulink documentation and search for legacy\_code.*

---

## 11 EHOOKS-DEV Reference Guide

### 11.1 EHOOKS-DEV Command Line Usage

The EHOOKS-DEV ECU Target Support tools allow the EHOOKS build process to be completely executed from the command line rather than via the EHOOKS-DEV graphical user interface.

```
toolchaindriver.exe --prjfile=<project-xml-file> [--basedir=<base-dir>]
[--a2lfile=<a2l-file>] [--hexfile=<hex-file>]
[--sixfile=<scoop-ix-file>]
[--userdeffile=<userdef-file>] [--filterfile=<filter-file>]
[--a2lfileout=<a2l-file-out>] [--hexfileout=<hex-file-out>]
[--make-support-file]
[--a2lname-prefix=<a2lname-prefix>] [--a2lname-suffix=<a2lname-suffix>]
[--otb-conversion-warn] [--logdir=<logging-dir>] [--password=<password>]
[--prebuild=<prebuild-script>] [--postbuild=<postbuild-script>]
[--consistency-warn] [--verbosity=<level>] [--errorlimit=<limit>] [--nobuild]
[--licmode=<license-mode>] [--help] [--version] [--merge-all]
[--compiler-name=<name> --compiler-gcc=<gcc> --compiler-objcopy=<objcopy>]
```

Where:

- <project-xml-file>** The path to the EHOOKS project file. This path must be specified on command line.
- <base-dir>** The path that is used as the base directory for all relatively paths elsewhere on the command line or within the project file. The base directory applies to any relative paths used except for the project file, where the current working directory is used instead. If the base directory is not specified then it defaults to the same directory where the project file is located.
- <a2l-file>** The path to the ECU's ASAM-MCD 2MC file for input. This path can also be specified in the project file.
- <hex-file>** The path to the ECU's binary image file for input. This path can also be specified in the project file.
- <scoop-ix-file>** The path to SCOOP-IX software interface file for input. This path can also be specified in the project file.
- <userdef-file>** The path to user provided data definitions file for input. This path can also be specified in the project file.
- <filter-file>** The path to measurement/process hook filter file. This path can also be specified in the project file.
- <a2l-file-out>** The path for the output ASAM-MCD 2MC file. This path can also be specified in the project file.
- <hex-file-out>** The path for the output binary (HEX) image file. This path can also be specified in the project file.
- <logging-dir>** The path to the tool-chain driver logging directory. Default is no logging.
- <prebuild-script>** The path of a pre-build script.
- <postbuild-script>** The path of a post-build script.
- <a2lname-prefix>** A valid C identifier used as a prefix for the A2L names of measurements and characteristics created on behalf of Simulink or ASCET (i.e. specified in the SCOOP-IX software interface definition file).
- <a2lname-suffix>** A valid C identifier used as a postfix for the A2L names of measurements and characteristics created on behalf of Simulink or ASCET (i.e. specified in the SCOOP-IX software interface definition file).
- <level>** The message verbosity level (0 = debug, 1 = verbose, 2 = info, 3 = warning, 4 = error, 5 = fatal). Default is 3.

- <limit>** The number of errors that can occur until the driver stops execution. Default is 1 (stop immediately on first error).
- <password>** The password used for ASAM-MCD 2MC Tier-1 IF\_DATA decryption. This option can also be specified in the project file. If no password is specified either on the command-line or within the project file, then it is assumed that the Tier-1 data is unencrypted.
- <license-mode>** The licensing mode to use - locked or unlocked. This option can also be specified in the project file. If no licensing mode is specified either on the command line or within the project file, then the default is locked.
- consistency-warn** Generate warnings instead of errors if inconsistencies are found in the input files.
- otb-conversion-warn** If it is not possible to generate a reversible conversion function for an OTB function input or output then instead of generating an error generate a warning and use the identity conversion.
- nobuild** Performs configuration consistency checking without building a binary (HEX) image.
- help** Displays help information on the command line usage
- version** Displays version information only
- make-support-file** This creates a file called EHOOKSSupport.7z that can be sent to your ETAS support team to help with technical support issues.
- merge-all** Allows merging of code, constant and data sections of hex files. If this is not specified the default behavior is to merge data sections only.

The following three arguments allow the user to specify an alternative compiler that will be used to compile the EHOOKS code. To use this option the named compiler must have been specified during prep and all three arguments must be specified on the command line (see section 11.5.3 for more detail.)

- compiler-name=** the compiler name as specified by the ECU SW provider.
- compiler-gcc=** the path to the gcc compiler (including the executable file name.)
- compiler-objcopy=** the path to the objcopy executable.

#### 11.1.1 Back-End Configuration File

It is possible to create an optional configuration file to apply build options for every build. This file should be named "ehooks.cfg" and should be placed in the Back-End install path. The configuration file specifies a set of build options which will be applied for any build.

The build options are stored in a file in the following format:

Each line in the configuration file contains one option only.

Lines starting with the character # are used to add comments and thus ignored.

Example:

```
# options file for project
--verbosity=1
--make-support-file
```

#### 11.1.2 Front-End Configuration File

It is possible to create an optional configuration file to contain the default options for the Front-End. This file should be named "ehooks-default.cfg" and should be placed in the Front-End install path. The configuration file specifies the default options used by the EHOOKS Front-End (as shown in the options dialog).

The default options are stored in a file in the following format:

Each line in the configuration file contains one option only.

Lines starting with the character # are used to add comments and thus ignored.

Example:

```
# options file for project
--verbosity=1
--make-support-file
```

## 11.2 EHOOKS-DEV Custom Build Steps

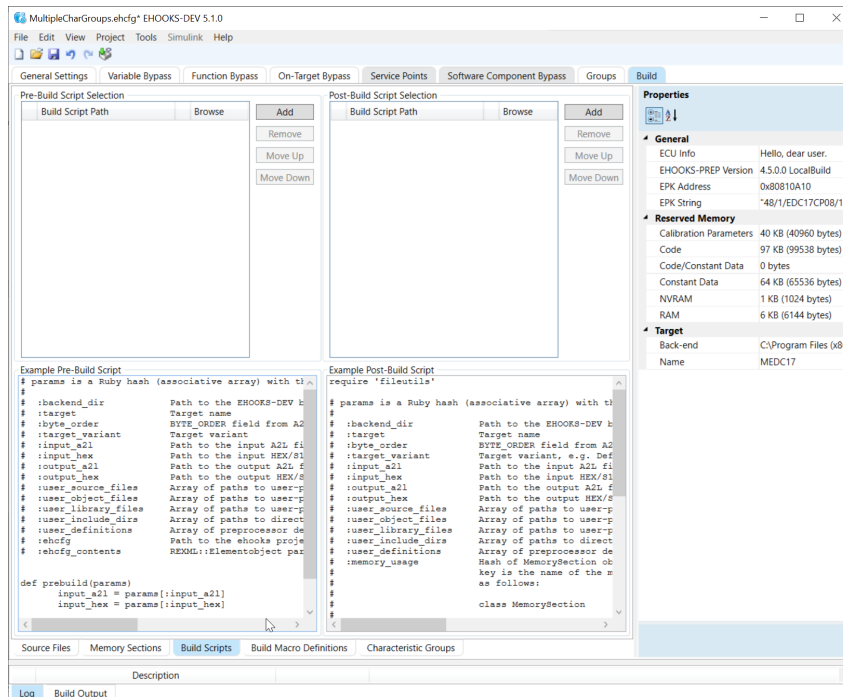
EHOOKS-DEV allows for custom build steps to be inserted into the EHOOKS build process by supporting the execution of Ruby scripts immediately before and after the hooked ECU files are built. There are 7 possible Ruby scripts that may be executed in the following order:

- `pregenerate_global.rb`
- Generation of EHOOKS code.
- `tier1_prebuild.rb`
- `prebuild_global.rb`
- `<project_prebuild>`
- Build of hooked ECU files.
- `<project_postbuild>`
- `postbuild_global.rb`
- `tier1_postbuild.rb`

The `tier1_prebuild.rb` and `tier1_postbuild.rb` scripts are provided by the Tier-1 and embedded in the A2L file. The EHOOKS-DEV user has no control over these scripts.

The `pregenerate_global.rb`, `prebuild_global.rb` and `postbuild_global.rb` scripts are contained in the `<install-dir>\Build` directory. These scripts are used for every build. These scripts may be used to run scripts that should run for every EHOOKS-DEV project.

`<project_prebuild>` and `<project_postbuild>` are project specific scripts specified in the EHOOKS-DEV GUI.



Prebuild and postbuild scripts are passed a Ruby hash as an argument. This contains information about the project. See the `prebuild_global.rb` and `postbuild_global.rb` scripts for information about the contents of the hashes.

Example: assume that one wants to run a tool called `chksumgen.exe` to update a checksum in a hooked `.hex/.s19` file. This could be done in either `global_postbuild.rb` or a project specific postbuild script. For example:

```
def postbuild(params)
  output_hex_file = params[:output_hex]
  chksumgen = "d:\\temp\\chksumgen.exe"
  puts "Postbuild script: #{chksumgen} #{output_hex_file}"
  system( "#{chksumgen} #{output_hex_file}" )
end
```

### 11.2.1 Pre-Generate Scripts

EHOOKS-DEV supports a `pregenerate_global.rb` script, which is executed at the start of the EHOOKS-DEV process, before the EHOOKS-generated source files are created.

This script may be used to run scripts that should run for every EHOOKS-DEV project.

The behavior of this script is the same as that of the pre- and post-build scripts, although the pre-generate script has access to fewer parameters. For a detailed description of the parameters available, see the `pregenerate_global.rb` script deployed with your EHOOKS-DEV Back-End installation.

## 11.3 EHOOKS-DEV Simulink Integration Scripting Interface

The EHOOKS-DEV Simulink Integration package comes with full support for scripting using Matlab M scripts. This allows complete control over the configuration of the EHOOKS block to be performed via this scripting interface.

### 11.3.1 Adding EHOOKS Blocks

EHOOKS blocks can be added to the model via the MATLAB `ADD_BLOCK` function. For details of this function see the MATLAB/Simulink user documentation or type `help add_block` <Enter> at the MATLAB command prompt. The `ADD_BLOCK` function requires the source and destination block path and can then take pairs of block properties and values for the block.

Table 3: EHOOKS Block Source Paths

EHOOKS Block =====	Block Source Path =====
Configuration Block	ehooks_lib/Library Blocks/Configuration
ECU Trigger Source Block	ehooks_lib/Library Blocks/ECU Trigger Source
ECU Variable Multiple Read Block	ehooks_lib/Library Blocks/ECU Variable Reads
ECU Variable Single Read Block	ehooks_lib/Library Blocks/ECU Variable Read
ECU Variable Multiple Write Block	ehooks_lib/Library Blocks/ECU Variable Writes
ECU Variable Single Write Block	ehooks_lib/Library Blocks/ECU Variable Write
ECU Backup Variable Multiple Read Block	ehooks_lib/Library Blocks/ECU Backup Variable Reads
ECU Backup Variable Single Read Block	ehooks_lib/Library Blocks/ECU Backup Variable Read
ECU Trigger Delegation Block	ehooks_lib/Library Blocks/ECU Trigger Delegate
ECU Hook Control Variable Write Block	ehooks_lib/Library Blocks/ECU Hook Control Variable Write
ECU Value Parameter Read Block	ehooks_lib/Library Blocks/ECU Value Parameter Read
ECU Function Call Block	ehooks_lib/Library Blocks/ECU Function Call



### 11.3.2 EHOOKS Simulink APIs

This section provides information on each of the API calls provided by the EHOOKS-DEV Simulink Integration Package, including the function names, parameters and examples of usage.

### 11.3.2.1 ehooks\_add\_port\_to\_msmtvariable\_block(block, argStruct)

Adds an ECU measurement variable read or write data port to a given ECU measurement variable multi-read/write block.

#### Syntax:

```
ehooks_add_port_to_msmtvariable_block(arg1, arg2)
```

#### Usage:

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_add_port_to_msmtvariable_block(block, argstruct)
```

Or:

```
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU Measurement Variable Reads', block)
or
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU Measurement Variable Writes', block)

ehooks_add_port_to_msmtvariable_block(handle, argstruct)
```

The input argument structure argstruct may contain the following fields:

Name	ASAM-MCD-2MC name of the measurement being read or written.
DisplayId	ASAM-MCD-2MC display ID of the measurement being read or written.
DataType	ASAM-MCD-2MC data type name of the measurement being read or written. Defaults to 'FLOAT64_IEEE'.
Conversion	Boolean specifying whether to perform data conversion. Defaults to true. DataType is set to 'double' in this case.
Index	Index into the array. Should only be used for array elements. For a scalar measurement, this argument shouldn't be specified.

'Name' is mandatory while all other fields are optional.

If the argument structure is an array with N number of arguments then N ports will get added to the block.

#### Usage examples:

Example 1 - adding a write to the scalar measurement ecu\_var\_x:

```
argstruct = struct('name', 'ecu_var_x', ...
    'datatype', 'SLONG', ...
    'conversion', false)

ehooks_add_port_to_msmtvariable_block(...
    'model/xyz/ECU_MsmtVariable_Writes', argstruct)
```

Example 2 - adding a write to array elements 1-3 of ecu\_array\_z:

```
argstruct(1) = struct('name', 'ecu_array_var_z', 'index', 1)
argstruct(2) = struct('name', 'ecu_array_var_z', 'index', 2)
argstruct(3) = struct('name', 'ecu_array_var_z', 'index', 3)

ehooks_add_port_to_msmtvariable_block(...
    'model/xyz/ECU_MsmtVariable_Writes', argstruct)
```

### 11.3.2.2 ehooks\_add\_to\_msmtvariable\_reads\_block(block, argStruct)

Adds an ECU measurement variable read data port to a given ECU measurement variable multiple read block.

#### Syntax:

```
ehooks_add_to_msmtvariable_reads_block(arg1, arg2)
```

#### Usage:

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_add_to_msmtvariable_reads_block(block, argstruct)
```

Or:

```
handle = add_block_slx('ehooks_lib/Library Blocks/ECU Measurement Variable Reads', block)
```

```
ehooks_add_to_msmtvariable_reads_block(handle, argstruct)
```

The input argument structure argstruct may contain the following fields:

Name	ASAM-MCD-2MC name of the measurement being read.
DisplayId	ASAM-MCD-2MC display ID of the measurement being read.
Conversion	Boolean specifying whether to perform data conversion.
Index	Index into the array. Should only be used for array elements. For a scalar measurement, this argument shouldn't be specified.

Name is mandatory while all other fields are optional.

#### Usage examples:

Example 1 - adding a read of the scalar measurement ecu\_var\_x:

```
argstruct = struct('name', 'ecu_var_x', ...
                  'conversion', false)

ehooks_add_to_msmtvariable_reads_block(...
    'model/xyz/ECU_MsmtVariable_Reads', argstruct)
```

Example 2 - adding a read of array element ecu\_array\_z[4]:

```
argstruct = struct('name', 'ecu_array_z', ...
                  'index', 4)

ehooks_add_to_msmtvariable_reads_block(...
    'model/xyz/ECU_MsmtVariable_Reads', argstruct)
```

### 11.3.2.3 ehooks\_add\_to\_msmtvariable\_writes\_block(block, argStruct)

Adds an ECU measurement variable write data port to a given ECU measurement variable multiple write block.

#### Syntax:

```
ehooks_add_to_msmtvariable_writes_block(arg1, arg2)
```

#### Usage:

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_add_to_msmtvariable_writes_block(block, argstruct)
```

Or:

```
handle = add_block_slx('ehooks_lib/Library Blocks/ECU_MsmtVariable_Writes', block)
```

```
ehooks_add_to_msmtvariable_writes_block(handle, argstruct)
```

The input argument structure argstruct can contain the following fields:

Name	ASAM-MCD-2MC name of the measurement being written.
DisplayId	ASAM-MCD-2MC display ID of the measurement being written.
Conversion	Boolean specifying whether to perform data conversion.
Index	Index into the array. Should only be used for array elements. For a scalar measurement, this argument shouldn't be specified.

Name is mandatory while all other fields are optional.

#### Usage examples:

Example 1 - adding a write to the scalar measurement ecu\_var\_x:

```
argstruct = struct('name', 'ecu_var_x', ...
                  'conversion', false)

ehooks_add_to_msmtvariable_writes_block(...
    'model/xyz/ECU_MsmtVariable_Writes', argstruct)
```

Example 2 - adding a write to array element ecu\_array\_z[4]:

```
argstruct = struct('name', 'ecu_array_z', ...
                  'index', 4)

ehooks_add_to_msmtvariable_writes_block(...
    'model/xyz/ECU_MsmtVariable_Writes', argstruct)
```

#### 11.3.2.4 `argStruct = ehooks_get_callable_function_properties(arg1, arg2)`

Gets the property values of an ECU callable function.

**Syntax:**

```
argstruct = ehooks_get_callable_function_properties(arg1, arg2)
```

**Usage:**

Given a string variable 'name' containing the name of an ECU trigger process:

```
argstruct = ehooks_get_callable_function_properties(name)
```

Or:

```
argstruct = ehooks_get_callable_function_properties(pid, name)
```

where pid is the identifier of the hook configuration project, determined via the function `ehooks_get_project_identifier()`. If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The function will return a structure containing the following fields:

**Usage examples:**

Example - getting the properties for ECU callable function 'ecu\_proc\_x'

```
var = ehooks_get_callable_function_properties('ecu_proc_x')
```

returns e.g.

```
var =
```

### 11.3.2.5 argStruct = ehooks\_get\_calparameter\_properties(arg1, arg2)

Gets the property values of an ECU calibration parameter (characteristic).

#### Syntax:

```
argstruct = ehooks_get_calparameter_properties(arg1, arg2)
```

#### Usage:

Given a string variable 'name' containing the name of an ECU calibration parameter:

```
argstruct = ehooks_get_calparameter_properties(name)
```

Or:

```
argstruct = ehooks_get_calparameter_properties(pid, name)
```

where pid is the identifier of the hook configuration project, determined via the function ehooks\_get\_project\_identifier(). If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The function will return a structure containing the following fields:

Name	A2L element name.
LongId	A2L long identifier.
DisplayId	A2L visible name (DISPLAY_IDENTIFIER).
DataType	A2L data type name.
Conversion	A2L conversion name (COMPU_METHOD name).
Deposit	A2L deposit name (RECORD_LAYOUT name).
BitMask	A2L bit mask integer value (BIT_MASK).
LowerLimit	A2L lower limit value.
UpperLimit	A2L upper limit value.
Type	A2L characteristic type [VALUE VAL_BLK MAP CURVE CUBOID ASCII].
MatrixDim	A2L sizes array for matrix dimensions X, Y and Z (MATRIX_DIM). Contains [ 0 0 0 ] for scalar parameters.

#### Usage examples:

Example - getting the properties for ECU parameter 'ecu\_param\_x':

```
argstruct = ehooks_get_calparameter_properties('ecu_param_x')
```

returns e.g.

```
argstruct =
    Name: 'ecu_param_x'
    LongId: 'ecu_param_x description'
    DisplayId: ''
    DataType: 'UBYTE'
    Conversion: 'OneToOne'
    Deposit: ''
    BitMask: 0
    MatrixDim: [ 0 0 0 ]
    UpperLimit: 255
    LowerLimit: 0
    Type: 'VALUE'
```

11.3.2.6 `argStruct = ehooks_get_complex_calparameter_properties(arg1, arg2)`

Gets the property values of an ECU calibration parameter (characteristic).

**Syntax:**

```
argstruct = ehooks_get_complex_calparameter_properties(arg1, arg2)
```

**Usage:**

Given a string variable 'name' containing the name of an ECU calibration parameter:

```
argstruct = ehooks_get_complex_calparameter_properties(name)
```

### 11.3.2.7 argStruct = ehooks\_get\_msmtvariable\_hook\_properties(arg1, arg2)

Gets the property values of an EHOOKS measurement variable hook.

#### Syntax:

```
argstruct = ehooks_get_msmtvariable_hook_properties(arg1, arg2)
```

#### Usage:

Given a string variable 'name' containing the name of an EHOOKS measurement variable hook created for writes to an ECU measurement variable:

```
argstruct = ehooks_get_msmtvariable_hook_properties(name)
```

Or:

```
argstruct = ehooks_get_msmtvariable_hook_properties(pid, name)
```

where pid is the identifier of the hook configuration project, determined via the function ehooks\_get\_project\_identifier(). If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The function will return a structure containing the following fields:

Enabler	Name of the enabler calibration parameter to create.
Indicator	Name of the indicator measurement variable to create.
Control	Name of the control variable to create.
RoutingMode	Signal routing mode (can be 'replacement', 'offset' or 'multiply').
Groups	Comma-separated list of groups to add the hook to.
BackupCopy	Backup copy option (can be 'default', 'enabled', or 'disabled').
SafetyCheck	Safety check option (can be 'default', 'enabled', or 'disabled').
ForcedWrites	Comma-separated list of processes that should forcedly trigger the variable hook.
InlineForcedWrites	Inline forced process writes option (can be 'enabled' or 'disabled').

#### Usage examples:

Example - getting the properties for variable hook 'ecu\_var\_x':

```
argstruct = ehooks_get_variable_hook_properties('ecu_var_x')
```

returns e.g.

```
argstruct =
  Enabler: 'EH_enabler_ecu_var_x'
  Indicator: 'EH_indicator_ecu_var_x'
  Control: 'EH_control_ecu_var_x'
  Groups: 'G1,G2'
  ForcedWrites: ''
  RoutingMode: 'replacement'
  BackupCopy: 'enabled'
  SafetyCode: 'disabled'
  InlineForcedWrites: 'disabled'
```



### 11.3.2.8 argStruct = ehooks\_get\_msmtvariable\_properties(arg1, arg2)

Gets the property values of an ECU measurement variable.

#### Syntax:

```
argstruct = ehooks_get_msmtvariable_properties(arg1, arg2)
```

#### Usage:

Given a string variable 'name' containing the name of an ECU measurement variable:

```
argstruct = ehooks_get_msmtvariable_properties(name)
```

Or:

```
argstruct = ehooks_get_msmtvariable_properties(pid, name)
```

where pid is the identifier of the hook configuration project, determined via the function ehooks\_get\_project\_identifier(). If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The function will return a structure containing the following fields:

Name	A2L name.
LongId	A2L long identifier.
DisplayId	A2L visible name (DISPLAY_IDENTIFIER).
DataType	A2L data type.
Conversion	A2L conversion name (COMPU_METHOD name).
LowerLimit	A2L value lower limit.
UpperLimit	A2L value upper limit.
Hookable	Boolean indicating whether the measurement variable can be hooked by EHOOKS or not.
Readable	Boolean indicating whether the measurement variable can be read by EHOOKS or not.

#### Usage examples:

Example - getting the properties for ECU variable 'ecu\_var\_x':

```
argstruct = ehooks_get_msmtvariable_properties('ecu_var_x')
```

returns e.g.

```
argstruct =
    Name: 'ecu_var_x'
    LongId: 'ecu_var_x description'
    DisplayId: ''
    Hookable: 1
    Readable: 1
    DataType: 'UBYTE'
    Conversion: 'OneToOne'
    UpperLimit: 255
    LowerLimit: 0
```

### 11.3.2.9 projectFilePath = ehooks\_get\_project\_file\_path(block)

Gets the current EHOOKS project file path used by a model.

**Syntax:**

```
arg = ehooks_get_project_file_path(arg1)
```

**Usage:**

Given a variable 'system' containing the model name:

```
projectFilePath = ehooks_get_project_file_path(system)
```

Or:

```
projectFilePath = ehooks_get_project_file_path()
```

will return the EHOOKS-DEV hook project file path, e.g. the string 'C:\temp\my\_system.ehcfg'.  
If no argument is passed then the current model (bdroot) is used.

#### 11.3.2.10 `projectId = ehooks_get_project_identifier(block)`

Gets the current EHOOKS project identifier used by a model.

**Syntax:**

```
arg = ehooks_get_project_identifier(arg1)
```

**Usage:**

Given a variable 'system' containing the model name:

```
projectId = ehooks_get_project_identifier(system)
```

Or:

```
projectId = ehooks_get_project_identifier()
```

will return the EHOOKS-DEV hook project identifier, e.g. the string '7296bb82-0185-4cc0-8e71-d6543c973402'. If no argument is passed then the current model (bdroot) is used.

### 11.3.2.11 argStruct = ehooks\_get\_project\_properties(arg1)

Gets the property values of an EHOOKS project.

#### Syntax:

```
argstruct = ehooks_get_project_properties(arg1)
```

#### Usage:

Given a string variable 'pid':

```
argstruct = ehooks_get_project_properties(pid)
```

Or:

```
argstruct = ehooks_get_project_properties()
```

where pid is the identifier of the hook configuration project, determined via the function ehooks\_get\_project\_identifier(). If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The function will return a structure containing the following fields:

ProjectId	Hook project identifier.
ProjectFile	Path to the hook project file.
InputA2LFile	Path to the input A2L file.
OutputA2LFile	Path to the output A2L file.
InputBinFile	Path to the input binary image file.
OutputBinFile	Path to the output binary image file.
BackendDir	Path to the back-end installation directory.
BackendTarget	Back-end target identifier.

#### Usage examples:

Example - getting the properties of the currently open project:

```
argstruct = ehooks_get_project_properties()
```

returns e.g.

```
argstruct =
    ProjectId: '7296bb82-0185-4cc0-8e71-d6543c973402'
    ProjectFile: 'C:\\temp\\project.ehcfg'
    InputA2LFile: 'C:\\temp\\ecu.a2l'
    OutputBinFile: 'C:\\temp\\ecu_hooked.a2l'
    InputBinFile: 'C:\\temp\\ecu.hex'
    OutputBinFile: 'C:\\temp\\ecu_hooked.hex'
    BackendDir: 'C:\\ETAS\\EHOOKS Back-Ends\\xyzSH2'
    BackendTarget: 'xyzSH2:Default'
```

### 11.3.2.12 `argStruct = ehooks_get_triggerprocess_properties(arg1, arg2)`

Gets the property values of an ECU trigger process.

**Syntax:**

```
argstruct = ehooks_get_triggerprocess_properties(arg1, arg2)
```

**Usage:**

Given a string variable 'name' containing the name of an ECU trigger process:

```
argstruct = ehooks_get_triggerprocess_properties(name)
```

Or:

```
argstruct = ehooks_get_triggerprocess_properties(pid, name)
```

where pid is the identifier of the hook configuration project, determined via the function `ehooks_get_project_identifier()`. If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The function will return a structure containing the following fields:

Name	Visible name of the dispatch process.
Period	Activation period of the trigger process in milliseconds (0 for asynchronous activation).

**Usage examples:**

Example - getting the properties for ECU process 'ecu\_proc\_x'

```
var = ehooks_get_triggerprocess_properties('ecu_proc_x')
```

returns e.g.

```
var =
    Name: 'ecu_proc_x'
    Period: 100
```

**11.3.2.13** `argStruct = ehooks_get_versions(arg1)`

Gets the versions of the EHOOKS-DEV Front-End, EHOOKS-DEV Back-End and the EHOOKS-Simulink integration package.

**Syntax:**

```
argstruct = ehooks_get_versions(arg1)
```

**Usage:**

Given a string variable 'pid':

```
argstruct = ehooks_get_versions(pid)
```

Or:

```
argstruct = ehooks_get_versions()
```

where pid is the identifier of the hook configuration project, determined via the function `ehooks_get_project_identifier()`. If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The function will return a structure containing the following fields:

<code>FrontendVersion</code>	Version of the EHOOKS-DEV Front-end
<code>BackendVersion</code>	Version of the EHOOKS-DEV Back-end for the current target, or an empty string if no back-end could be found.

**Usage examples:**

Example - getting the properties of the currently open project:

```
argstruct = ehooks_get_versions()
```

returns e.g.

```
argstruct =
    FrontendVersion: '3.0.0.42'
    BackendVersion: '3.0.0.42'
```

#### 11.3.2.14 `variableNames = ehooks_lookup_display_identifier(arg1, arg2)`

Gets the ECU variables names for a specified A2L display identifier.

**Syntax:**

```
retval = ehooks_lookup_display_identifier(arg1, arg2)
```

**Usage:**

Given a string variable 'dispName' containing the an ECU measurement variable display identifier:

```
variableNames = ehooks_lookup_display_identifier(dispName)
```

Or:

```
variableNames = ehooks_lookup_display_identifier(pid, dispName)
```

where pid is the identifier of the hook configuration project, determined via the function `ehooks_get_project_identifier()`. If the project identifier is not passed then the identifier for the current model (bdroot) is used.

Will return an array of identifiers of measurements whose display identifier is 'ecu\_var\_x\_disp'.

**Usage examples:**

Example - getting the variable names for a display identifier 'ecu\_var\_x\_disp':

```
variableNames = ehooks_lookup_display_identifier('ecu_var_x_disp')
```

returns e.g.

```
variableNames = { 'ecu_var_1', 'ecu_var_2' }
```

### 11.3.2.15 ehooks\_open\_configuration\_block(block, visible)

Opens an EHOOKS configuration block (i.e. opens the attached EHOOKS project).

**Syntax:**

```
ehooks_open_configuration_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_open_configuration_block(block, visible)
```

Or:

```
handle = add_block_slx('ehooks_lib/Configuration', block)
```

```
ehooks_open_configuration_block(handle, visible);
```

where visible is an integer value specifying the EHOOKS-DEV front-end visibility:

visible is 0: Explicitly hide the EHOOKS-DEV GUI.

visible is 1: Explicitly show the EHOOKS-DEV GUI.



### 11.3.2.16 ehooks\_open\_frontend(projectFilePath, visibility)

Starts the EHOOKS-DEV configuration tool with the specified EHOOKS project configuration file.

**Syntax:**

```
ehooks_open_frontend(arg1, arg2)
```

**Usage:**

Given a string variable 'filePath' containing the path an EHOOKS configuration file (e.g. 'c:\temp\model.ehcfg'):

```
ehooks_open_frontend(filePath)
```

Or:

```
ehooks_open_frontend(filePath, visibility)
```

where visible is an integer value specifying the EHOOKS-DEV front-end visibility:

visibility is 0: Explicitly hide the EHOOKS-DEV GUI.

visibility is 1: Explicitly show the EHOOKS-DEV GUI.

visibility has other value or is absent: Don't explicitly hide or show.

### 11.3.2.17 updated = ehooks\_set\_msmtvariable\_hook\_properties(arg1, arg2, arg3)

Sets the property values of an EHOOKS measurement variable hook.

#### Syntax:

```
ehooks_set_msmtvariable_hook_properties(arg1, arg2, arg3)
```

#### Usage:

Given a string variable 'name' containing the name of an EHOOKS measurement variable hook created for writes to an ECU measurement variable:

```
ehooks_set_msmtvariable_hook_properties(name, argstruct)
```

Or:

```
ehooks_set_msmtvariable_hook_properties(pid, name, argstruct)
```

where pid is the identifier of the hook configuration project, determined via the function ehooks\_get\_project\_identifier(). If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The input argument argstruct may contain the following fields:

Enabler	Name of the enabler calibration parameter to create.
Indicator	Name of the indicator measurement variable to create.
Control	Name of the control variable to create.
RoutingMode	Signal routing mode (can be 'replacement', 'offset' or 'multiply').
Groups	Comma-separated list of groups to add the hook to.
BackupCopy	Backup copy option (can be 'default', 'enabled', or 'disabled').
SafetyCheck	Safety check option (can be 'default', 'enabled', or 'disabled').
ForcedWrites	Comma-separated list of processes that should forcedly trigger the variable hook.
InlineForcedWrites	Inline forced process writes option (can be 'enabled' or 'disabled').

All fields are optional.

#### Usage examples:

Example - defining an enabler parameter for variable hook 'ecu\_var\_x':

```
argstruct = struct('enabler', 'ecu_var_x.enabler');
ehooks_set_variable_hook_properties('ecu_var_x', argstruct);
```

### 11.3.2.18 ehooks\_set\_project\_properties(arg1, arg2)

Sets the property values of an EHOOKS project.

#### Syntax:

```
ehooks_set_project_properties(arg1, arg2)
```

#### Usage:

```
ehooks_set_project_properties(argstruct)
```

Or:

```
ehooks_set_project_properties(pid, argstruct)
```

where pid is the identifier of the hook configuration project, determined via the function ehooks\_get\_project\_identifier(). If the project identifier is not passed then the identifier for the current model (bdroot) is used.

The input argument argstruct may contain the following fields:

InputA2lFile	Path to the A2L file to import.
OutputA2lFile	Path to the A2L file to create.
InputBinFile	Path to the binary image file to import.
OutputBinFile	Path to the binary image file to create.
DecryptPassword	Password used to decrypt the input A2L file.
RememberPassword	True or false - should the password be stored in the ehcfg file? Default is true.
LicenseMode	Licensing mode used, 'locked' or 'unlocked'.
CreateBackupCopies	true or false
CreateSafetyChecks	true or false
Version	String to place in the project version field.
Description	String to place in the project description field.
GlobalEnabler	Name of the parameter to create for the global enabler. No enabler is created if missing.
GlobalIndicator	Name of the measurement to create for the global indicator. No indicator is created if missing.

Note that all settings beside the current A2L input file path are cleared if the corresponding structure fields are empty or absent.

#### Usage examples:

Example 1:

```
argstruct = struct('decryptpassword', 'abc123', ...
                  'rememberpassword', 0, ...
                  'inputa2lfile', 'c:\\input.a2l', ...
                  'outputa2lfile', 'c:\\output.a2l', ...
                  'inputbinfile', 'c:\\input.hex', ...
                  'outputbinfile', 'c:\\output.hex');

ehooks_set_project_properties(argstruct);
```

Example 2:

```
argstruct = struct('licensingmode', 'unlocked', ...
                  'createbackupcopies', true, ...
                  'createsafetychecks', false, ...
```

```
'globalenabler', 'GlobalEnabler', ...  
'globalindicator', 'GlobalIndicator', ...  
'version', 'Version 3.14159', ...  
'description', 'Hello world');
```

```
ehooks_set_project_properties(argstruct);
```

#### 11.3.2.19 ehooks\_update\_complexcalparam\_block(block, argStruct)

Updates the (port) parameters of a given ECU Complex Calibration block

**Syntax:**

```
ehooks_update_complexcalparam_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_complexcalparam_block(block, argstruct)
```

### 11.3.2.20 ehooks\_update\_configuration\_block(block, argStruct)

Updates the properties of an EHOOKS configuration block.

**Syntax:**

```
ehooks_update_configuration_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_configuration_block(block, argstruct)
```

Or:

```
handle = add_block_slx('ehooks_lib/Configuration', block)
```

```
ehooks_update_configuration_block(handle, argstruct)
```

The input argument argstruct may contain the following fields:

ProjectFile	Path of the EHOOKS configuration project file to use.
UseFloatTypedPorts	Whether floating-point data type shall always be used for the ECU measurement variable read/write block ports. Ensures backwards compatibility with EHOOKS V2.0. The default value is false.
UseDisplayIds	Whether to use ASAM-MCD-2MC display identifiers instead of ASAM-MCD-2MC names for the generation of port labels. The default value is false.

**Usage examples:**

Example - using a non-default project file path 'c:\temp\my\_config.ehcfg':

```
argstruct = struct('ProjectFile', 'c:\\temp\\my_config.ehcfg', ...
                  'UseFloatTypedPorts', false);

ehooks_update_configuration_block(argstruct);
```

**11.3.2.21 ehooks\_update\_ecu\_call\_block(block, argStruct)**

Updates the properties of an ECU function-call block.

**Syntax:**

```
ehooks_update_ecu_call_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_ecu_call_block(block, argstruct)
```

Or:

```
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU_Function_Call', block)

ehooks_update_ecu_call_block(handle, argstruct)
```

The input argument argstruct may contain the following fields:

Process	Name of the process that will act as dispatch point.
---------	--

**Usage examples:**

Example - setting the enabler name, the ECU dispatch process name

and enabling forced writes:

```
argstruct = struct(...
    'Process', 'ecu_proc')

ehooks_update_ecu_call_block(...
    'my_system/OTB1SubSys/FuncCall', argstruct)
```

### 11.3.2.22 ehooks\_update\_fcncalltrigger\_delegate\_block(block, argStruct)

Updates the properties of an ECU function-call trigger delegation block.

**Syntax:**

```
ehooks_update_fcncalltrigger_delegate_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_fcncalltrigger_delegate_block(block, argstruct)
```

Or:

```
handle = add_block(
    'ehooks_lib_slx/Library Blocks/ECU_FcnCallTrigger_Delegate', block)

ehooks_update_fcncalltrigger_delegate_block(handle, argstruct)
```

The input argument argstruct may contain the following fields:

FunctionToCall	Name of the function-call trigger source block (implemented EHOOKS OTB function) that is manually dispatched.
----------------	---

'FunctionToCall' is mandatory while all other fields are optional.

**Usage examples:**

Example - setting the OTB function to manually call:

```
argstruct = struct('FunctionToCall', 'OTB2')

ehooks_update_fcncalltrigger_delegate_block(...
    'my_system/OTB1SubSys/TrigDelegate', argstruct)
```



## 11.3.2.23 ehooks\_update\_fcncalltrigger\_source\_block(block, argStruct)

Updates the properties of an ECU function-call trigger source block.

**Syntax:**

```
ehooks_update_fcncalltrigger_source_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_fcncalltrigger_source_block(block, argstruct)
```

Or:

```
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU_FcnCallTrigger_Source', block)

ehooks_update_fcncalltrigger_source_block(handle, argstruct)
```

The input argument argstruct may contain the following fields:

Enabler	Name of the enabler characteristic to create.
Indicator	Name of the indicator measurement to create.
Control	Name of the control variable to create.
Groups	Comma-separated list of groups to add the hook to.
Process	Name of the process that will act as dispatch point.
Period	The bypass container's period in milliseconds, or 0 if it is not periodic.
ForceWrites	Set the flag specifying whether the OTB function's outputs should have forced writes at its dispatch point. 0 = false, 1 = true.

'Process' is mandatory while all other fields are optional.

**Usage examples:**

Example - setting the enabler name, the ECU dispatch process name

and enabling forced writes:

```
argstruct = struct(...
    'Enabler', 'OTB1.enabler', ...
    'Process', 'ecu_proc', ...
    'ForceWrites', 1)

ehooks_update_fcncalltrigger_source_block(...
    'my_system/OTB1SubSys/TrigSource', argstruct)
```

### 11.3.2.24 ehooks\_update\_hookctrlvariable\_write\_block(block, argStruct)

Updates the properties of an EHOOKS hook control variable write block.

**Syntax:**

```
ehooks_update_hookctrlvariable_write_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_hookctrlvariable_write_block(block, argstruct)
```

Or:

```
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU_HookCtrlVariable_Write', block)

ehooks_update_hookctrlvariable_write_block(handle, argstruct)
```

The input argument argstruct may contain the following fields:

Variable	Name of the created EHOOKS hook control variable (measurement) that is written.
----------	---

'Variable' is mandatory while all other fields are optional.

**Usage examples:**

Example - setting the control variable name:

```
argstruct = struct('Variable', 'my_hook_control_var')

ehooks_update_hookctrlvariable_write_block(...
    'my_system/OTB1SubSys/ECU_HookCtrlVariable_write', argstruct)
```

### 11.3.2.25 ehooks\_update\_msmtvariable\_block(block, argStruct)

Updates the (port) parameters of a given ECU measurement variable single or multi read/write block. If a multi read/write block is passed then the data port configuration gets reset to a single port having property values assigned according to the arguments passed to this function.

#### Syntax:

```
ehooks_update_msmtvariable_block(arg1, arg2)
```

#### Usage:

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_msmtvariable_block(block, argstruct)
```

Or:

```
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU Measurement Variable Read', block)
or
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU Measurement Variable Write', block)

ehooks_update_msmtvariable_block(handle, argstruct)
```

The input argument argstruct may contain the following fields:

Name	ASAM-MCD-2MC name of the measurement being read or written.
DisplayId	ASAM-MCD-2MC display ID of the measurement being read or written.
DataType	ASAM-MCD-2MC data type name of the measurement being read or written. Defaults to 'FLOAT64_IEEE'.
Conversion	Boolean specifying whether to perform data conversion. Defaults to true. DataType is set to 'double' in this case.
Index	Index into the array. Should only be used for array elements. For a scalar measurement, this argument shouldn't be specified.
StorageReuse	Boolean specifying whether to allow reuse of the data port storage. Defaults to false.

'Name' is mandatory while all other fields are optional.

#### Usage examples:

Example 1 - setting up a write to the scalar measurement ecu\_var\_x:

```
argstruct = struct('name', 'ecu_var_x', ...
    'datatype', 'SLONG', ...
    'conversion', false)

ehooks_update_msmtvariable_block(...
    'model/xyz/ECU_MsmtVariable_Write', argstruct)
```

Example 2 - setting up a write to array element ecu\_array\_z[4]:

```
argstruct = struct('name', 'ecu_array_var_z', ...
    'index', 4)

ehooks_update_msmtvariable_block(...
    'model/xyz/ECU_MsmtVariable_Writs', argstruct)
```

### 11.3.2.26 ehooks\_update\_scalarcalparam\_read\_block(block, argStruct)

Updates the properties of an ECU scalar parameter read block.

#### Syntax:

```
ehooks_update_scalarcalparam_read_block(arg1, arg2)
```

#### Usage:

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_scalarcalparam_read_block(block, argstruct)
```

Or:

```
handle = add_block(...
    'ehooks_lib_slx/Library Blocks/ECU_ScalarCalParam_Read', block)

ehooks_update_scalarcalparam_read_block(handle, argstruct)
```

The input argument argstruct may contain the following fields:

Parameter	Name of the scalar ECU calibration parameter (characteristic) that is read.
DisplayId	Display name of the characteristic being read.
Conversion	Boolean specifying whether to perform type conversion.
DataType	ASAM-MCD-2MC data type name of the measurement being read or written. Defaults to 'FLOAT64_IEEE'.
'Parameter' is mandatory while all other fields are optional.	

#### Usage examples:

Example - setting the ECU parameter name:

```
argstruct = struct('Parameter', 'ecu_param_x')

ehooks_update_scalarcalparam_read_block(...
    'my_system/OTB1SubSys/ECU_ScalarParamRead', argstruct)
```

### 11.3.2.27 ehooks\_update\_valueblockcalparam\_block(block, argStruct)

Updates the (port) parameters of a given ECU Value Block Calibration block

**Syntax:**

```
ehooks_update_valueblockcalparam_block(arg1, arg2)
```

**Usage:**

Given a string variable 'block' containing the path to the EHOOKS block:

```
ehooks_update_valueblockcalparam_block(block, argstruct)
```

### 11.3.2.28 paramAdded = irt\_register\_custom\_parameter(argStruct)

Appends a parameter to the global IRT parameter registry.

#### Usage:

To register a parameter:

```
irt_register_custom_parameter(argStruct)
```

The argument argStruct may contain the following fields:

For characteristics of all types (type == 'value|value\_blk|map|curve') generically:

```
'type'          string('value') ['value', 'value_blk', 'curve', 'map']
'usage'         string('')      ['values', 'axis'; for type=='value_blk']
'identifier'    string('')
'model_id'     string('')
'model_name'   string('')
'description'  string('')
'phys_min'     double(-1.7e+305)
'phys_max'     double(+1.7e+305)
'resolution'   uint32(1)
'unit'         string('')
'index_mode'   string('column_dir') ['column_dir', 'row_dir']
```

For look-up table characteristics (type == 'map' || type == 'curve') additionally:

```
'x_axis'       string('') [for type=='curve' || type=='map']
'y_axis'       string('') [for type=='map']
```

'identifier' and 'type' are mandatory while all other fields are optional.

Example - adding a read of the scalar measurement ecu\_var\_x

```
argstruct = struct('identifier', 'NC_CBK_EX_NR', 'type', 'value',
                  'phys_min', -100, 'phys_max', 100);
irt_register_custom_parameter(argstruct);
```

### 11.3.2.29 paramAdded = irt\_register\_custom\_signal(argStruct)

Appends a global signal (variable) to the global IRT signal registry.

#### **Usage:**

To register a signal:

```
irt_register_custom_signal(argStruct)
```

The argument argStruct may contain the following fields:

```
'identifier'    string('')
'model_id'     string('')
'model_name'   string('')
'description'  string('')
'phys_min'     double(-1.7e+305)
'phys_max'     double(+1.7e+305)
'unit'         string('')
```

'name' is mandatory while all other fields are optional.

Example - adding a read of the scalar measurement ecu\_var\_x

```
argstruct = struct('identifier', 'signal1', 'model_name', 'prefix.signal1',
                  'phys_min', -100, 'phys_max', 100);
irt_register_custom_signal(argstruct);
```

## 11.4 Special Purpose RAM

It is possible for the Tier-1 to reserve sections of RAM for special purposes. The EHOOKS-DEV user might want (for example) to place specific variables in non-volatile RAM where the value of the variable must be retained when the ECU is powered off. Each such special RAM section has an associated regular expression. The Tier-1 must provide the EHOOKS-DEV user with information on the special purpose RAM available and the regular expressions used.

When EHOOKS is assigning a variable (also called a measurement or signal) it has created to RAM it compares the name of the variable to the regular expressions associated with special purpose RAM sections. If the name of the variable matches a regular expression, the variable is located in the corresponding RAM section. If the variable does not match any regular expressions it is located in ordinary RAM.

For example:

Consider three RAM sections: two special purpose RAM sections NVRAM and ECC, and ordinary RAM C

RAM	Regular Expression
NVRAM	"_NVRAM\$"
ECC	"^ECC_"
Ordinary RAM	-

The following variables will be located in the sections indicated:

Variable Name	Location
MyVariable_NVRAM	NVRAM
MyVariable	Ordinary RAM
ECC_Variable	ECC

Variables located in special purpose RAM are not automatically initialised by EHOOKS (either zeroed or given an initialisation value) since this would not make sense for non-volatile RAM.

## 11.5 Advanced Project Options

### 11.5.1 Cached Register Warning Message

This section provides an explanation of the cached register warning message:

"Any variables calculated from the value of X may not use the hooked value, as the original ECU-calculated value is cached in a register. If you don't see the hooked value being used, you may also need to hook these additional variables. See the EHOOKS-DEV User Guide for more details."

If you see the cached register warning and also get unexpected hook behaviour you probably need to change your hook configuration. For example, assume that the ECU contains program code like:

Assign V to X

Assign F(X) to Y



Where X and Y are variables, V is a value (e.g. another variable, a calibration parameter or a constant), and F() is a function. If you get the cached register warning for a hook of X and when you run your bypass experiment it appears that the bypass value for X is not being used in the calculation of Y then you would need to hook Y as well as X and generate a bypass value for Y.

To understand this warning message it is necessary to consider how an optimizing compiler converts program code (hand written or generated by a tool like ASCET or MATLAB/Simulink) into the instructions executed by the processor in the ECU. Assume that the ECU software contains some program code that does the following:

Program Code Example A:

Assign V to X

Assign V to Y

The compiler may convert these assignments into a sequence of processor instructions something like:

Instruction Sequence A:

1. Load V into processor register R
2. Store register R in X
3. Store register R in Y

Now consider some different program code:

Program Code Example B:

Assign V to X

Assign X to Y

One may expect a compiler to convert this program code into a sequence of instructions something like:

Instruction Sequence B1:

1. Load V into processor register R
2. Store register R in X
3. Load X into processor register R
4. Store register R in Y

However, an optimizing compiler will notice that in step 2 it stores the value in register R into X and therefore step 3 is redundant. The compiler will therefore omit step 3 and generate the instruction sequence:

Instruction Sequence B2:

1. Load V into processor register R
2. Store register R in X
3. Store register R in Y

Note that instruction sequence B2 is the same as instruction sequence A. That is, although program code example A is different to program code example B the sequence of processor instructions generated is the same.

EHOOKS is only able to analyse the processor instructions for ECU software (i.e. the hex or s19 file). Therefore in the above example EHOOKS cannot tell if it is analysing program code example A or program code example B. When EHOOKS hooks a variable it modifies the memory cell containing the variable. If EHOOKS is told to hook variable X and is given the instruction sequence A/B2 it will effectively modify the sequence as follows:

1. Load V into the processor register R
2. Store bypass value in X
3. Store register R in Y

If the instruction sequence was the result of compiling program code example A then this would result in the expected behaviour. However, if the instruction sequence was the result of compiling program code example B then this would not produce the correct behaviour because one would expect the bypass value for X to be stored in Y as well. If EHOOKS were to store the bypass value in register R as well as X then one would get the expected behaviour for program code example B but not for program code example A. If EHOOKS could tell which program code had been used it could take the correct approach. Unfortunately the optimizing compiler has discarded the information that would allow EHOOKS to tell which program code had been used.

The above examples are deliberately very simple to illustrate the problem. Real ECU program code is much more complex, but the same principal applies, an optimizing compiler removes redundant loads of variables and unfortunately EHOOKS cannot tell that this has happened. EHOOKS attempts to identify instruction sequences where the compiler **might** have removed redundant loads and generates the cached register warning. However EHOOKS can only identify an instruction sequence like A/B2, it cannot tell whether it comes from program code like example A or from program code like example B.

### 11.5.2 Overriding Cached Registers with EHOOKS

If a Cached Register warning message, as described in the previous section, is generated for a specific hook and that hook also exhibits unexpected behaviour, it might be useful under some circumstances to apply an override to the hook. This override can be achieved in EHOOKS by optionally updating the value stored in the source register in addition to updating the value stored in the memory address of the hooked variable.



#### **WARNING**

*The use of the Cached Register Override feature in EHOOKS should be used with extreme caution. Updating the source register for a hooked variable could also modify the value of other, unhooked variables. This option should therefore only be enabled when the EHOOKS build log warns about register caching **and** the hook is not behaving as expected.*

---

The cached register override feature can be accessed from the EHOOKS Project menu as shown in figure 11.1:

From the **Project** menu select **Advanced** and then click on **Cached Register Overrides**.

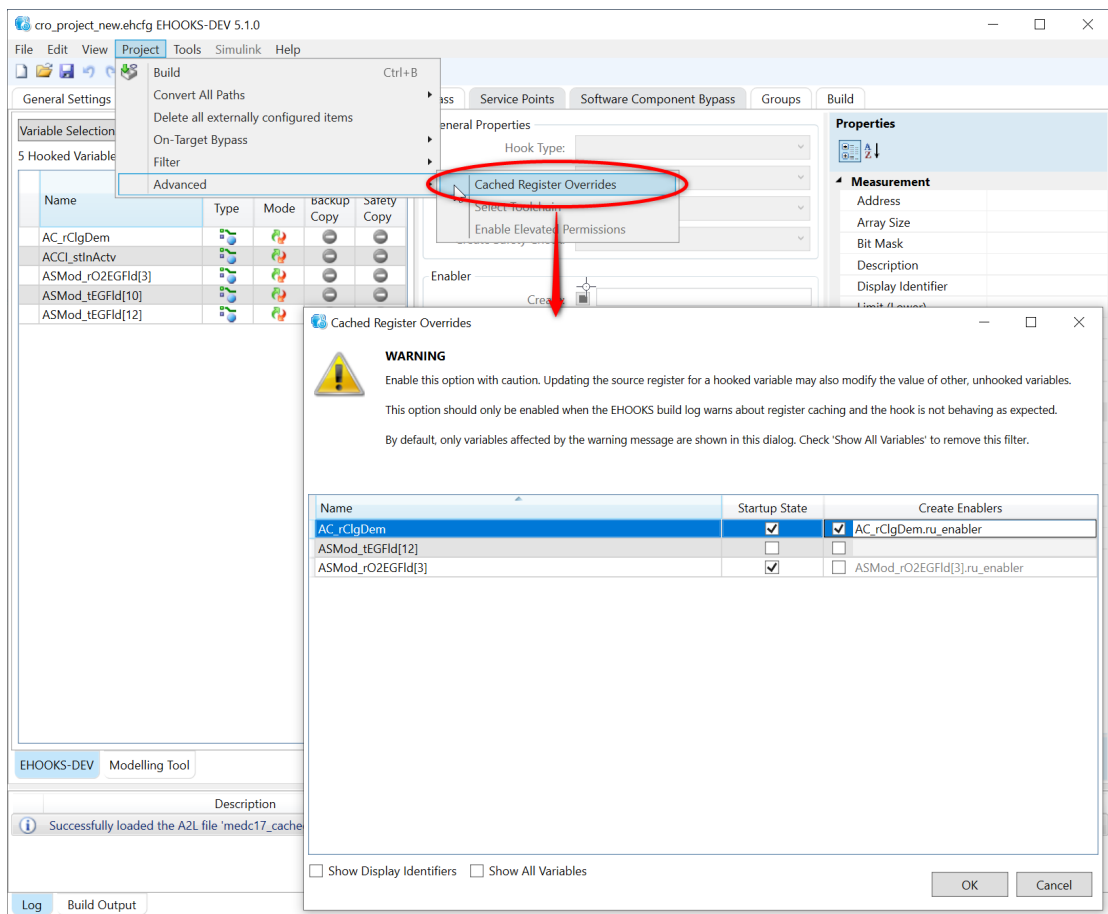


Figure 11.1: Cached Register Overrides in EHOOKS

This brings up a dialog listing all hooked variables with two checkboxes that allow updates to the source register to be configured. The first of these checkboxes is labelled 'Startup State'. When this checkbox is ticked the register update will be enabled at ECU startup. The second checkbox is labelled 'Create Enablers'. If this checkbox is ticked EHOOKS will create a new enabler characteristic that allows the register update to be turned on or off at runtime using INCA.

In the example shown in figure 11.1, the following behaviours have been configured:

- 1) **Variable AC\_trqDes:** Neither checkbox is ticked. No register > updates will be performed for this hooked variable.
- 2) **Variable Brk\_stRed:** Only the 'Startup State' checkbox is ticked. > Register updates will be enabled at ECU startup, but they cannot > be controlled at runtime using INCA.
- 3) **Variable CEngDsT\_dt:** Only the 'Create Enablers' checkbox is > ticked. An enabler called CEngDsT\_dt.ru\_enabler is created that > allows the register updates to be turned on or off at runtime > using INCA. At ECU startup, the register updates are disabled.
- 4) **Variable DFES\_ctEntry:** Both the 'Startup State' and 'Create > Enablers' checkboxes are ticked. An enabler called > DFES\_ctEntry.ru\_enabler is created that allows the register > updates to be turned on or off at runtime using INCA. At ECU > startup, the register updates are enabled.

### 11.5.3 Building EHOOKS Code with an Alternative Compiler

EHOOKS-DEV uses a GCC toolchain when compiling C code generated by EHOOKS for integration into the ECU software. This toolchain is distributed along with the EHOOKS-DEV Back-End.

During ECU software preparation the Tier-1 software provider may specify alternative compilers (and compiler options) that can be used for the EHOOKS build. Information about the permitted alternative compilers will be embedded in the prepared A2L file during ECU software preparation.

One of the permitted compilers may be selected by name via a menu item in the EHOOKS-DEV front-end or via command line arguments to the toolchain-driver (described above in 11.1).

In the EHOOKS-Dev Front-End navigate to the **Project** menu select **Advanced** and then click on **Select Toolchain** (figure 11.2). The option will be enabled if alternative compilers are permitted. Selection is then via a dialog-box (figure 11.3) which will also prompt the user to locate the gcc compiler and objcopy executables on the local machine. The name and selected paths are saved as part of the DEV project.

---

#### NOTICE

*EHOOKS always requires gcc, math and C standard libraries (libc.a, libm.a and libgcc.a) So, in the linking stage EHOOKS will point to these libraries inside the installed version of EHOOKS-DEV directory. If the users want to provide their own libraries, they can do this by using the Build tab in EHOOKS-DEV Front-End (see 5.8.1 Configuring Build Source Files).*

---

**WARNING**

The user is responsible for ensuring that the appropriate compiler executables are selected. Ehooks is tested with the default compiler and options and therefore changes to the compiler and options are not guaranteed by ETAS to work. When using an alternative compiler, in-case of problems related to the build or problems that do not occur with the default compiler and options, please seek support from the Tier-1 software provider before contacting the Ehooks support team.

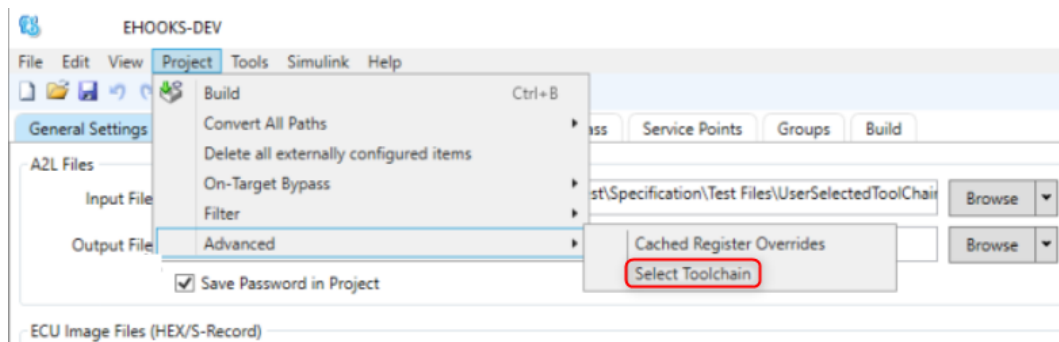


Figure 11.2: Alternative compilers menu location.

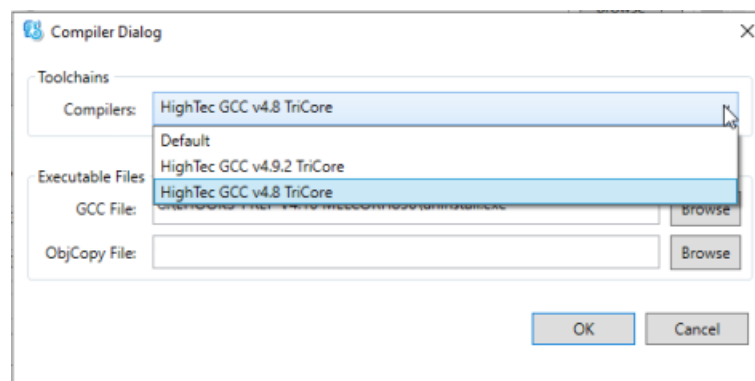


Figure 11.3: Alternative compilers selection dialog.

#### 11.5.4 Enable Elevated Permissions

EHOOKS allows Forced Writes with Elevated Permissions and can be used if the Tier-1 has supported it. Elevated Permissions may be needed when the bypass container doesn't have the correct memory access privileges to perform the forced write. The majority of cases will not need to use Elevated Permissions.

To enable Elevated Permissions, in the EHOOKS-Dev Front-End navigate to **Project** menu, select **Advanced** and then click on **Enable Elevated Permissions**. A warning will appear regarding the feature overriding memory protection, which needs to be accepted to continue.

With the option selected, Variable Bypass hooks and On-Target Bypass functions can be configured to use Elevated Permissions, as described in sections [5.2.2 Configuring Properties of a Variable Hook](#) and [Configuring Properties of an On-Target Bypass Function](#).

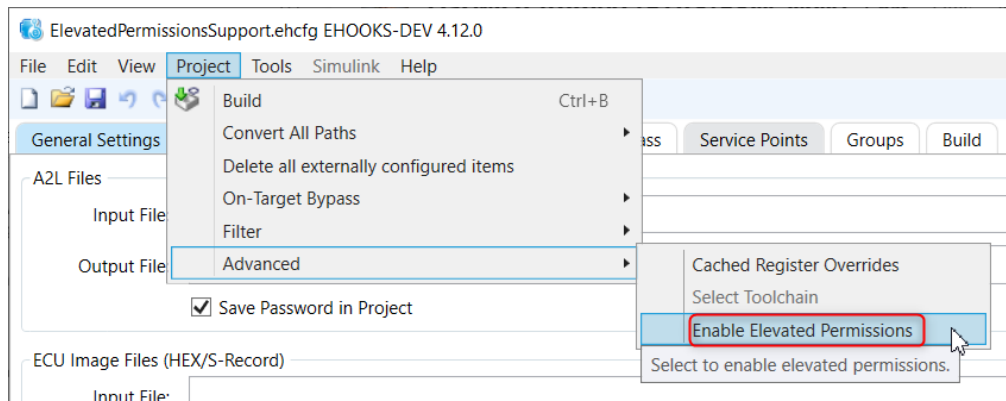


Figure 11.4: Elevated Permissions menu location.

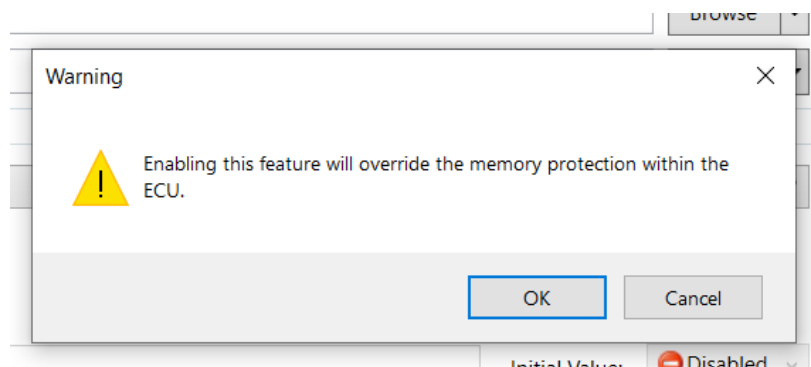


Figure 11.5: Elevated Permissions warning dialog.

## 11.6 Variable Initialization

When working with EHOOKS-DEV, you may need to define variables that are explicitly initialized. These could be hook control variables or variables defined in on-target bypass functions for example. Since the ECU startup code doesn't know about these variables, EHOOKS provides an initialization function called `EH_InitRam` for this purpose.

As part of the EHOOKS ECU SW preparation process, your ECU supplier may have provided processes in which this initialization function can be run. If this is not the case, and there are variables that need to be initialized, you will see the following warning:

```
“WARNING: Data initialization is required but no initialization processes exist.
Initialisation must be performed from an on-target bypass function in order for data
to be initialized properly. See the EHOOKS-DEV User Guide for more details.”
```

In this case, the initialization function must be called manually from an on-target bypass function in order for data to be initialized properly. The initialization function is defined as below:

```
EH_OPTIMAL_RETURN EH_InitRam(EH_ULONG maxBytesToCopy)
```

The argument `maxBytesToCopy` tells this function how many bytes of initialization data to copy per-call. If this argument is less than the number of bytes of data that need initializing, the function needs to be called in a bypass function that runs enough times to completely initialize all data. If the argument passed is 0, a default value will be used.

The function returns a code indicating whether initialization has completed. As it is not safe for a bypass function using the initialized values to run before the initialization has completed, the bypass function needs to check whether initialization has completed before running. Therefore, the function should be called as below:

```
EH_USER_BYPASS_FUNC(my_bypass_function)
{
    // Initializes some data and returns
    // if the initialization does not complete
    if (!EH_InitRam(0))
    {
        return 0;
    }
    // Bypass function body
    ...
    // End of function
    return result;
}
```

## 11.7 EHOOKS-DEV File Formats

### 11.7.1 EHOOKS-DEV Project Configuration File

Each EHOOKS-DEV project configuration file must contain a target identifier indicating the EHOOKS-DEV Back-End target to be used.

This identifier is in the form:

```
<Hardware identifier="<target>:<variant>" name="<target>" vendor="<vendor>" />
```

where:

- <target> is the name of the EHOOKS ECU target
- <variant> is the name of the ECU variant used (or Default if the ECU has no variants)
- <vendor> is the name of the organization that supplies the ECU.

The EHOOKS-DEV Front-End will automatically generate the correct target identifier in the project configuration file when an A2L file is loaded.

The EHOOKS-DEV Project Configuration XML file format is fully documented in <install-dir>\Schemas\OEMProjectDoc.zip as part of the EHOOKS product installation.

### 11.7.2 EHOOKS-DEV User Definition Files

The EHOOKS-DEV User Definition XML file format is fully documented in <install-dir>\Schemas\OEMUserDefinesDoc.zip as part of the EHOOKS product installation.

### 11.7.3 EHOOKS-DEV Filter File

The EHOOKS-DEV project filter file is a simple ASCII file format containing the names of ECU variables and processes that EHOOKS-DEV should not allow for hooking. Each ECU variable/process must be separated by whitespace or end-of-line tokens. Hash (#) tokens can be used to include comments in the file, a hash token causes all characters from the hash token to the end-of-line token to be ignored.

Source Code 7: Example Project Filter File

```
# Filter file for project Wibble

# Prohibited measurements are as follows:
AirCtl_dmAirDesMax_mp           # Comment Text
PFltP0p_stEngP0p
InjCrv_qPiI1Des AirCtl_mDesBasEOM0 # Two on one line!

# Prohibited processes:
Proc_1
Proc_2
```



## 12 EHOOKS Limitations

Although powerful, there are some things that EHOOKS is not able to do. This section describes some situations where EHOOKS might have shortcomings.

### 12.1 Non-Atomic Writes

A non-atomic write is encountered when the instruction writing to a hooked variable does not match the size of the hooked variable. There are two variations:

1. The instruction targets a data type smaller than the variable; and
2. The instruction targets a data type larger than the variable.

The first case is typically seen when a block-copy function is used to write to a variable (e.g. the C library's `memcpy` function). In this situation, the block-copy function is likely to write to memory one byte at a time in a loop. Storing data to variables larger than a byte will therefore fail. This can typically be seen when structures are copied, and hooking of individual elements of the structure is attempted.

The second case is seen in some compiler optimizations, where writes to consecutive address locations are combined into a single store instruction – e.g. writing four consecutive byte variables with a single 32-bit store.

In both cases, EHOOKS will recognize that a hooked variable is being written to but, because the size of the write does not match the size of the variable, no hook will be placed. This usually causes EHOOKS-DEV to emit a warning to the user.

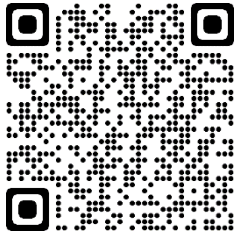
### 12.2 Timing

One of the powerful features of EHOOKS is the ability to add new code (OTB) to the ECU software. Where time-critical operations are taking place, it must be remembered that adding new code may break the timing properties of the ECU code. This is especially relevant when using dynamic function bypass where it is possible to run 'old' and 'new' code side by side in order to compare the results of a new algorithm for example.

## 13 Contact Information

### 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/hotlines](http://www.etas.com/hotlines)



### ETAS Headquarters

ETAS GmbH

Borsigstraße 24  
70469 Stuttgart  
Germany

Phone: +49 711 3423-0  
Fax: +49 711 3423-2106  
Internet: [www.etas.com](http://www.etas.com)