
ASCET V5.2

Getting Started

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 2007** ETAS GmbH, Stuttgart

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

The name INTECRIO is a registered trademark of ETAS GmbH.

Document EC010010 R5.2.2 EN

TTN F 00K 103 222

Contents

1	Introduction	7
1.1	System Information	7
1.2	User Information	8
1.2.1	User Profile	8
1.2.2	Manual Structure	8
1.2.3	How to Use this Manual	12
2	Program Installation	15
2.1	Preparation	15
2.1.1	Contents	15
2.1.2	System Requirements	15
2.1.3	Required User Privileges for Installation and Operation	16
2.2	Installation	17
2.2.1	Initial Installation	18
2.2.2	Special Installation Steps and Dialogs	25
2.3	Network Installation	29
2.3.1	Providing Data in the Network	29
2.3.2	Customizing the Network Installation	30
2.3.3	Installing ASCET from the Network Drive	33
2.4	Uninstalling ASCET	34

2.4.1	Automatic Uninstall	34
2.4.2	Custom Uninstall	36
3	Licensing	39
3.1	Obtaining Licenses	39
3.2	Licensing Status	42
3.3	Borrowing Licenses	43
4	Understanding ASCET	47
4.1	Increasing Efficiency in Control Unit Development	47
4.1.1	Modern Embedded Control Systems: Technical Mission	47
4.1.2	Development Processes: Economic Challenge	52
4.1.3	Innovative Technologies - Technological Visions.	54
4.2	Continuous Support for Embedded Control Systems	58
4.2.1	Entry-level Technology Bypass.	59
4.2.2	Prototyping	60
4.2.3	Automatic Code Generation.	60
4.2.4	Other Application Options for ETAS Development Tools	62
4.2.5	Interfaces and Standards in the Tool Chain	63
4.3	ASCET Development Environment in Practise	64
4.3.1	Physical Specification of Control Systems	66
4.3.2	Implementation and Code Generation	70
4.3.3	Prototyping with ASCET	74
4.3.4	Bypass	76
4.3.5	Reuse and Open Interfaces.	77
4.4	ASCET Software Structure	78
5	General Operation of ASCET	81
5.1	Window Structure	82
5.2	Button Bars	83
5.2.1	Buttons in the Component Manager	83
5.2.2	Button Bars in the Block Diagram Editor	84
5.2.3	Button Bars in the C Code and ESDL Editor.	87
5.2.4	Button Bars in the CT Block Editors.	87
5.2.5	Button Bar Elements in the Project Editor	89
5.2.6	Button Bar Elements in the Offline Experiment	91
5.3	Operation Using The Keyboard	92
5.3.1	General Keyboard Control	92
5.3.2	Keyboard Control According to the Windows Conventions.	93
5.4	Operation Using The Mouse	94
5.4.1	Drag & Drop.	95

5.5	Hierarchy Trees	95
5.6	Supporting Functions	97
5.6.1	Monitor Window	97
5.6.2	Keyboard Assignment	97
5.6.3	Manual and Online Help	97
6	Tutorial	99
6.1	A Simple Block Diagram	99
6.1.1	Preparatory steps	99
6.1.2	Specifying a Class	103
6.1.3	Summary	114
6.2	Experimenting with Components	115
6.2.1	Starting the Experimentation Environment	115
6.2.2	Setting up the Experimentation Environment	116
6.2.3	Using the Experimentation Environment	121
6.2.4	Summary	124
6.3	To Specify a Reusable Component	124
6.3.1	Creating the Diagram	125
6.3.2	Experimenting with the Integrator	133
6.3.3	Summary	137
6.4	A Practical Example	137
6.4.1	Specifying the controller	137
6.4.2	Experimenting with the Controller	141
6.4.3	A Project	142
6.4.4	To set up the Project	143
6.4.5	Experimenting with the Project	146
6.4.6	Summary	148
6.5	Extending the Project	148
6.5.1	Specifying the Signal Converter	148
6.5.2	Experimenting with the Signal Converter	151
6.5.3	Integrating the Signal Converter into the Project	154
6.5.4	Summary	158
6.6	Modeling a Continuous Time System	158
6.6.1	Motion Equation	159
6.6.2	Model Design	160
6.6.3	Summary	167
6.7	A Process Model	168
6.7.1	Specifying the Process Model	168
6.7.2	Integrating the Process Model	173
6.7.3	Summary	178

6.8	State Machines	178
6.8.1	Specifying the State Machine	179
6.8.2	How a State Machine Works	187
6.8.3	Experimenting with the State Machine	189
6.8.4	Integrating the State Machine in the Controller	190
6.8.5	Summary	192
6.9	Hierarchical State Machines	192
6.9.1	Specifying the State Machine	192
6.9.2	Experimenting with the Hierarchical State Machine	200
6.9.3	How Hierarchical State Machines Work	201
6.9.4	Summary	202
7	Glossary	203
7.1	Abbreviations	203
7.2	Terms	204
8	Reference Lists	215
8.1	Troubleshooting and User Feedback	215
8.2	ASCET Directories	217
8.2.1	Default Storage Directories	217
8.2.2	Changing Default Directories	218
8.3	Keyboard Control	219
8.3.1	General Control Functions	219
8.3.2	Keyboard Commands in the Component Manager	220
8.3.3	Keyboard Commands in the Monitor Window	220
8.3.4	Keyboard Commands in the Editors	221
8.3.5	Keyboard Commands in the Offline Experiment Environment	222
8.3.6	Measure and Calibration Windows in General	222
9	Windows XP Firewall and ASCET	227
9.1	Users with Administrator Privileges	228
9.2	Users without Administrator Privileges	231
9.3	Support and Problem Reporting	232
10	ETAS Contact Addresses	233
	Index	235

1 Introduction

ASCET provides an innovative solution for the functional and software development of modern embedded software systems. ASCET supports every step of the development process with a new approach to modelling, code generation and simulation, thus making higher quality, shorter innovation cycles and cost reductions a reality.

This manual supports the reader in getting to know ASCET, and quickly achieving results. It provides a step-by-step introduction to the system, while at the same time making all information easily accessible for reference.

1.1 System Information

The ASCET product family consists of a number of products that provide interfaces to simulation processors, third-party software packages and for remote access to ASCET. The following products are available for the current version of ASCET:

- *ASCET-MD*—support for the development and simulation of models.
- *ASCET-RP*—support for experimental targets to allow hardware-in-the-loop simulation and rapid prototyping applications. A toolbox for running ETk Bypass experiments is also integrated. ASCET-RP provides the connection to INTECRIO.
- *ASCET-SE*—support for various microcontroller targets. Generation of optimized executable code, including operating system configuration and integration, for various microcontrollers and two real-time operating systems.

Various kinds of additional modules are optional:

- *Configuration management*—provides an interface to configuration management tools.
- *ASCET-MIP*—This MATLAB integration package comes as a bundle that provides two different means for accessing Matlab software. The first part is an interface to the Matlab engine which allows you to couple ASCET and MATLAB at simulation level. The second part is a model converter that enables you to read Simulink models in ASCET.
- *ASCET-DIFF*—A comparison tool for ASCET models.

Various additional customer-specific products can be integrated in ASCET. More detailed information is available upon request.

1.2 User Information

1.2.1 User Profile

This manual addresses qualified personnel working in the fields of automobile control unit development and calibration. Specialized knowledge in the areas of measurement and control unit technology is required.

ASCET users should be familiar with the Microsoft Windows 98, Windows NT 4.0, Windows 2000, or Windows XP, operating system. All users should be able to execute menu commands, enable buttons, etc. Furthermore, the users should be familiar with the Windows file storage system, especially the connections between files and directories. The users have to know how to use the basic functions of the Windows File Manager and Program Manager or the Windows Explorer, respectively. Moreover, the users should be familiar with the "drag-and-drop" functionality.

Any user who is not familiar with the basic techniques found in Microsoft Windows should learn them before using ASCET. For more information on the Windows operating system, please refer to the manuals published by Microsoft Corporation.

Knowledge of a programming language, preferably ANSI C or Java, can be helpful for advanced users.

1.2.2 Manual Structure

The ASCET manual consists of three volumes:

1. Volume "ASCET V5.2 – Getting Started"
This volume provides you with basic information on the ASCET working principles.
2. Volume "ASCET V5.2 – User's Guide"
This volume describes the operation of ASCET-MD.
3. Volume "ASCET V5.2 – Reference Guide"
This volume contains a detailed description of the ASCET modeling language as well as numerous reference lists.

The contents of the individual volumes are described below in more detail.

Volume "ASCET V5.2 – Getting Started"

This volume contains the following chapters:

- **"Introduction"** (this chapter)
This chapter provides an outline of the possible applications of ASCET. Furthermore, it contains general information such as innovations in ASCET V5.2, user and system information.
- **"Program Installation"**
This chapter targets both all users who install, maintain or uninstall ASCET on a PC or a network as well as system administrators who provide ASCET on a file server so that the program can be installed via the network. It contains important information on the scope of delivery, hardware and software requirements for stand-alone and network installations and the preparation required for installation. The chapter also describes the procedures used to install and uninstall ASCET.
- **"Licensing"**
This chapter contains various licensing information, e.g. how to obtain a license file or how to borrow a license.
- **"Understanding ASCET"**
This chapter provides an overview of the ASCET system, the development process supported by it, and its place in the ETAS tool chain. This chapter should be read first by all users new to ASCET.
- **"General Operation of ASCET"**
This chapter provides information on the window and menu structures of ASCET, as well as control options using the mouse and the keyboard.
- **"Tutorial"**
The "Tutorial" mainly addresses users who are new to ASCET. It describes the use of ASCET using practice-oriented examples. The entire tutorial contents are subdivided into short individual components based on each other. Before you start working on the tutorial, you should have read chapter "Understanding ASCET" on page 47.

Note

ETAS offers efficient training in the use of ASCET in order to provide an even more thorough knowledge of ASCET, especially if the user has to gain a comprehensive insight in the functionality of ASCET in a very short period of time.

- **"Glossary"**
This chapter explains all technical terms used in the manual. The terms are listed in alphabetic order.

- **"Reference Lists"**

This chapter contains information on troubleshooting, the directory structure, and the reference files required. This chapter also includes a list of all keyboard commands sorted by working windows.

Volume "ASCET V5.2 – User's Guide"

This volume comprehensively describes all components of the ASCET system and provides detailed instructions on using them. Before you start reading this part, you should become acquainted with the ASCET software, i.e., work through the chapters "Understanding ASCET" and "Tutorial" of the ASCET Getting Started volume.

The description of the ASCET system is organized to reflect the chronological order of the development process of an embedded control system in ASCET.

This volume contains the following chapters:

- **"Introduction"**

Description of the typical workflow

- **"The Component Manager"**

Getting familiar with the user interface, controls and menu options, customizing ASCET, learning to use the Component Manager.

This chapter is relevant to all users of ASCET.

- **"Adding User-Defined Functions"**

This chapter describes how users can define own menu functions in various ASCET windows.

- **"Specification of Components and Projects"**

This chapter explains how to work with the different specification editors and how to change the attributes of the elements.

- **"Signals and Icons"**

This chapter describes how to manage and integrate signals and icons in the database.

- **"Experimentation"**

This chapter describes how to work with the experimentation environment during offline experiments, and gives an overview over the various measurement and calibration windows.

Online experiments, as well as experimenting with INCA or INTECRIO, are described in the ASCET-RP user's guide.

- **"Automatic Documentation"**

This chapter describes the automatic generation of documentation for ASCET components and projects.

Volume "ASCET V5.2 – Reference Guide"

The first part, "The Modeling Language", is a comprehensive reference to the various ways of describing embedded software systems in ASCET. It is advisable to work through the tutorial before reading any of the chapters of this part.

The following chapters belong to this part:

- **"Projects"**

The specification of an embedded control system is called a project. The structure of a project is described here.

- **"Components"**

Components are the building blocks of an embedded system. The various kinds of components are introduced in this chapter.

- **"Types and Elements"**

This chapter describes the kinds of variables and the data types supported by ASCET.

- **"Data and Implementations"**

Each variable in ASCET has data and an implementation, which are discussed here.

- **"Body Specification in ESDL"**

This chapter is about specifying components in ESDL, the model description language of ASCET.

- **"Body Specification with Block Diagrams"**

In this chapter the specification of components as block diagrams is described.

- **"Body Specification in C"**

Components can also be specified in C, which is discussed here.

- **"Continuous Time Systems"**

An overview of continuous time systems in ASCET, which are used to build mathematical models of technical processes.

- **"Continuous Time Basic Blocks"**

Basic blocks describe individual components of a continuous time system.

- **"Continuous Time Structure Blocks and Graphical Hierarchies"**

Structure blocks integrate the basic blocks into complete models.

- **"Projects and Hybrid Projects"**

In hybrid projects continuous time systems can be run alongside embedded system specifications.

The second part, "Reference Lists", describes the ASCET system library, and other reference information.

This part consists of the following chapters:

- **"The ASCET System Library"**

This chapter provides a detailed description of each component of the system library.

- **"Troubleshooting"**

Here, common user errors and known problems are listed together with advice on how to solve them.

- **"Code Generation Messages"**

This chapter contains the error messages that may appear during code generation, together with advice on how to adapt the software model in ASCET to avoid such errors.

1.2.3 How to Use this Manual

Documentation Conventions

All actions to be performed by the user are presented in a task-oriented format as illustrated in the following example. A *task* in this manual is a sequence of actions that have to be performed in order to achieve a certain goal. The title of a task description usually introduces the result of the actions, e.g. "To create a new component", or "To rename an element". Task descriptions often contain illustrations of the particular ASCET window or dialog box the task relates to.

To achieve a goal:

Any preliminary information...

- **Step 1**

Explanations are given underneath an action.

- **Step 2**

Any explanation for Step 2...

- Step 3

Any explanation for Step 3...

Any concluding remarks...

Specific example:

To create a new file:

When creating a new file, no other file may be open.

- Choose **File** → **New**.
The "Create file" dialog box is displayed.
- In the "File name" field, type the name of the new file.
The file name must not exceed 8 characters.
- Click **OK**.

The new file will be created and saved under the name you specified. You can now work with the file.

Typographic Conventions

The following typographic conventions are used in this manual:

Choose **File** → **Open**.

Menu commands are shown in **blue boldface**.

Click **OK**.

Buttons are shown in **blue boldface**.

Press <ENTER>.

Keyboard commands are shown in angled brackets and capitals.

The "Open File" dialog window opens.

Names of program windows, dialog boxes, fields, etc. are shown in quotation marks.

Select the file `setup.exe`.

Text in drop-down lists on the screen, program code, as well as path- and file names are shown in the `Courier` font.

A *distribution* is always a one-dimensional table of sample points.

General emphasis and new terms are set in *italics*.

The OSEK group (see <http://www.osekvd.org/>) has developed certain standards.

Links to internet documents are set in blue underlined font.

Important notes for the users are presented as follows:

Note _____

Important note for users.

2 Program Installation

The chapter entitled "Program Installation" targets both all users who install ASCET on a PC or a network, or maintain and uninstall the program as well as system administrators who provide ASCET on a file server so that the program can be installed via the network. It contains important information on the scope of delivery, hardware and software requirements for stand-alone and network installations and the preparation required for installation. The chapter also describes the procedures used to install and uninstall ASCET.

2.1 Preparation

Check the items supplied for completeness and your computer for compliance with the system requirements. Depending on the operating system used and the network connection, you have to make sure that you have the user privileges required.

2.1.1 Contents

ASCET is supplied with the following:

- ASCET CD ROM
 - ASCET program files
 - ASCET manuals and ETAS hardware documentation in PDF format (Acrobat Reader)
 - Manual "FLEXnet Licensing End User Guide" in PDF format
 - Acrobat Reader program files
- "ASCET Getting Started" manual (ASCET-MD only)

2.1.2 System Requirements

The following system requirements have to be met:

- 1 GHz Pentium PC (recommended: 2 GHz)
- WINDOWS® 2000, WINDOWS® XP
- 512 MB RAM (recommended: 512 MB)
- Hard disk with a minimum of 1 GB of free space (not including space for program data; recommended: > 1 GB)
- CD ROM drive
- VGA graphics card with VGA monitor and a resolution of at least 800 x 600 with 256 colors

2.1.3 Required User Privileges for Installation and Operation

User privileges required for installation:

In order to install ASCET on a PC, you need the user privileges of an administrator. Please contact your system administrator, if necessary.

User privileges required for operation (under WIN 2000/XP):

In order to operate ASCET under Windows 2000/XP, each user must receive the privilege called "Increase Scheduling Priority" from the administrator. This can be set using the User Manager.

Note

You need administrator rights to perform the settings described below.

Recommendation: Assign the privilege "Increase Scheduling Priority" to the local "User" group. To do so, proceed as follows:

To assign WIN 2000 – User Privilege "Increase Scheduling Priority":

- From the Windows Start Menu, choose **Settings** → **Control Panel** → **Administrative Tools** → **Local Security Policy**.
- Under **Local Policies** → **User Rights Assignment** activate **Increase scheduling priority** by double-clicking on it.
- Click on the **Add** button.
- Select the local workstation.
- Assign the "Increase Scheduling Priority" privilege to the `User` group by double-clicking on it.
- Confirm by clicking the **OK** button.
- Close the "Local Security Policy" window by clicking **OK**.
- Exit the local security policy settings.

To assign WIN XP – User Privilege "Increase Scheduling Priority":

- From the Start Menu, choose **Settings** → **Control Panel** → **Administrative Tools** → **Local Security Policy**.

- Under **Local Policies** → **User Rights Assignment** activate **Increase Scheduling Priority** by double-clicking on it.
The "Increase scheduling priority Properties" window is displayed.
- Click on the **Add User or Group** button.
The "Select Users or Groups" window is displayed.
- Click on the **Locations** button.
The "Locations" window is displayed.
- Select the local workstation and close the "Locations" window by clicking **OK**.
- In the "Select Users or Groups" window, click on the **Advanced** button to enable the automatic search feature.
- Click on the **Find now** button to display the list of users registered for the local workstation.
- In the "Name (RDN)" column, choose the names of the users or groups to whom you want to assign the privilege to increase the scheduling priority.
- Confirm by clicking the **OK** button.
- Close the "Select Users or Groups" window by clicking **OK**.
- Close the "Increase scheduling priority Properties" window by clicking **OK**.
- Exit the local security policy settings.

2.2 Installation

To install one of the products ASCET-MD, ASCET-RP, or ASCET-SE (see chapter 4.4 on page 78), in any case you have to install the ASCET base system first. This chapter describes the installation of the base system and ASCET-MD. The installation of ASCET-RP and ASCET-SE is described in the respective manuals.

Installation is performed in the same way if you install ASCET from the CD or the network drive.

Special issues to be observed during the installation (e.g., "Canceling the Installation" or "Overwriting an Existing Program Version") are described in chapter 2.2.2 on page 25.

2.2.1 Initial Installation

ASCET Base System

To start the ASCET installation:

- From the Start menu, select **Run**.
- In the command line, enter the path to the installation file (e.g. if installed from the CD: d:\ASCET5.2\ascet.exe).
- Confirm by clicking **OK**.

The installation program is started.

Follow the instructions:

- In the "EULA" window, activate the **Accept** option to accept the license agreement.
- Click **OK**.
- Follow the instructions displayed on the screen.

The **Next** button accepts your settings and proceeds to the next window, the **Back** button returns to the previous window, and **Cancel** aborts the installation.

To register ASCET:

- In the registration window, enter your personal information.

Registration information

Please enter user information in the fields below:

First name: Last name:

Company: Department:

Phone:

E-mail:

Street:

ZIP code: City:

Country:

< Back Next > Cancel

- Click on the **Next** button.

To specify ASCET paths:

When you have registered ASCET, you will be prompted to specify target directories for the data. This is done in two separate windows:

Select destination directories

Please select a directory for the ASCET files:

c:\ETAS\ASCET%2 Browse...

Current free disk space: 1633976 k
Free disk space after install: 1343602 k

Please select a directory for the ASCET data files:

c:\ETASData\ASCET%2 Browse...

Current free disk space: 1633976 k
Free disk space after install: 1343602 k

< Back Next > Cancel

Program *files* and program *data* are stored in different directories. When you uninstall or update the program later, only the program files will be deleted or overwritten. The program data will continue to be available to you. The program data includes the following:

- Databases
- User profiles

Note

You can install ASCET in a directory with blanks in its path. Before you do so, however, make sure that all external tools used with ASCET support path names with blanks, too.

- If you want to change the default directories, click on the **Browse** button.
- In the dialog box, select the desired directory. If you specify a directory which does not exist, the installation routine will automatically create it.
- Click on the **Next** button. The "Select global directories" window opens. Here, you can specify paths for log files and temporary files. These paths are used by all ETAS products.
- Use the **Browse** buttons to adjust the paths settings according to your wishes.
- Click on the **Next** button.

Settings for common files:

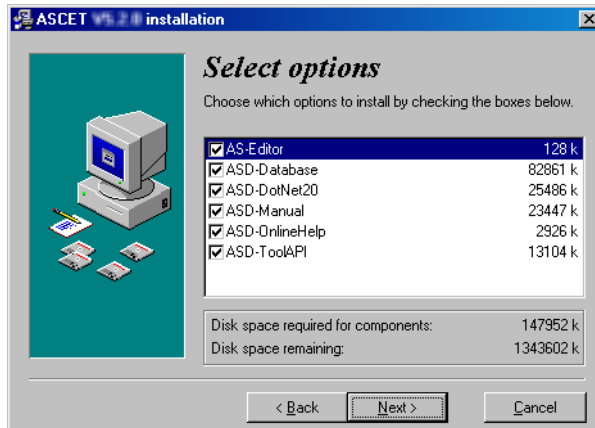
In the "Select Handling of ETAS-Shared modules" window, you specify the way the modules used in principle by all ETAS products are installed.

- Activate the **share modules between products** option when the modules are to be conjointly used. This is reasonable when you intend to use several ETAS products simultaneously.
- Activate the **use local copies for each product** option when ASCET shall be installed with its own copies of the modules, e.g., when you intend to adjust the components.

- Click on the **Browse** button to adjust the path settings for the modules.
- Click on the **Next** button.

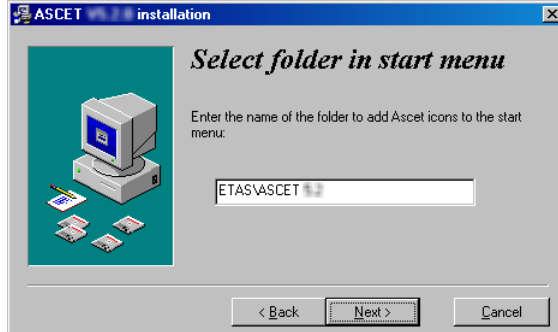
To specify the ASCET functional scope:

In the "Select options" window, you can specify the functional scope of ASCET.



- Mark the modules you want to install.
The following options can be selected:
 - *AS-Editor* – the arithmetic services editor.
 - *ASD-Database* – installs the ETAS system library and tutorial database in the export directory of your ASCET installation.
 - *ASD-Manual* – installs the ASCET manuals as PDF files in the ETAS\ETASManuals directory.
 - *ASD-OnlineHelp* – the ASCET online help is placed in the ETAS\ASCET5.2\help directory.
 - *ASD-ToolAPI* – installs the ASCET Automation interface.
- Click on the **Next** button.

To specify the ASCET folder in the Start menu:



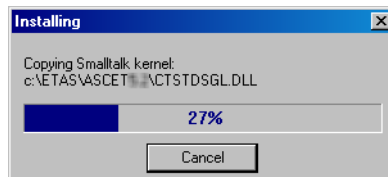
- Accept the default folder name
or
- specify a different folder name.
- Click on the **Next** button.

Installing ASCET:

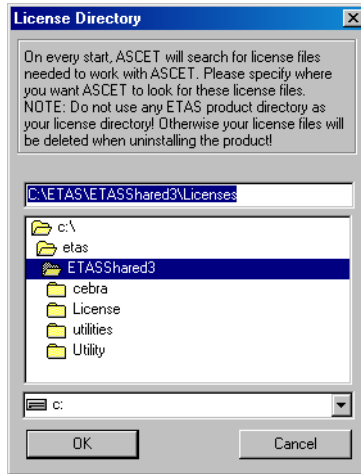
Note

With the next step, you start the installation.

- In the "Ready for installation" window, click **Next** to start the installation.
The program files will be copied. A bar chart indicates the copy progress.



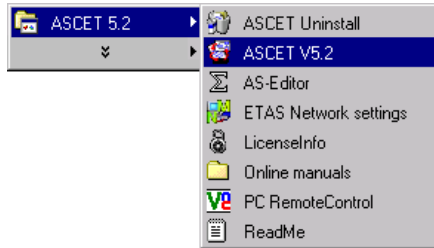
When all files are copied, the "License Directory " selection window opens.



Specifying the license file location:

- In the "License Directory" window, enter the directory where the license file will be stored. Each time ASCET needs a license, this directory is searched.
- Click **OK** to confirm your selection. The installation continues, and finally the "Installation complete" window appears.
- In the "Installation complete" window, click on the **Finish** button to finish the installation.

After restarting your PC, you will find the folder name you specified with the following entries in the Start menu:



- **ASCET Uninstall**
Starts the uninstall routine (see chapter 2.4).
- **ASCET V5.2**
Starts the ASCET program.
- **AS Editor**
Starts the AS Editor (see chapter 4.14 in the ASCET user's guide).
- **ETAS Network settings**
Starts the assistant for the configuration of the ETAS network.
- **LicenseInfo**
Starts the "Obtain License Info" window (cf. chapter 3).
- **Online manuals**
If you installed the online manuals, you can here open the `ETAS\ETASManuals` directory. Here, the manuals are stored in several subdirectories.
- **PC RemoteControl**
Enables the configuration of the remote interface.
- **ReadMe**
Provides current information on ASCET V5.2.

ASCET-MD

After installing the basic system, you can install ASCET-MD.

To install ASCET-MD:

- From the Start menu, select **Run**.

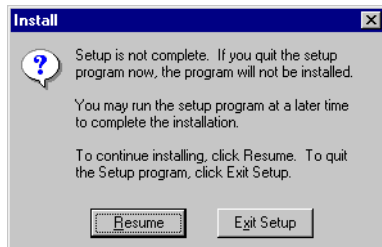
- In the command line, enter the path to the installation file (e.g. if installed from the CD: `d:\ASCET-MD V5.2\ASCET-MD.exe`).
- Confirm by clicking **OK**.
The installation program is started. Since the ASCET base system is already installed, you do not have to select path names, functional scope, etc.
- Click on **Next** to proceed to the next installation window.
- Click on **Back** to return to the previous installation window.
- Click on **Cancel** to abort the installation.

2.2.2 Special Installation Steps and Dialogs

To cancel the installation:

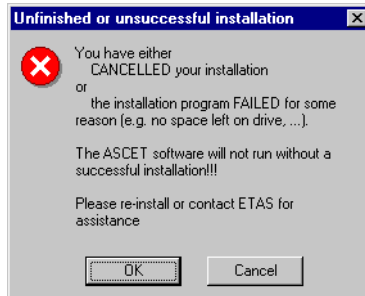
At this time during the installation process, you can still cancel the installation prematurely. Proceed as follows:

- Click the **Cancel** button in the current window.



- Click **Resume** to return to the previous dialog box.

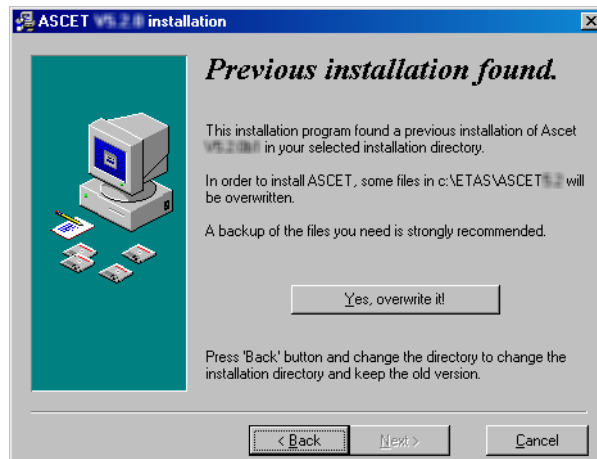
- Click the **Exit Setup** button to quit the setup program.



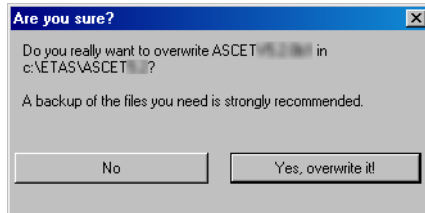
- Confirm that you want to cancel the installation by clicking **OK**.

To overwrite an existing program version:

If an older version of the software to be installed is found on the target workstation, or if an entirely different software exists in the selected installation directory, a corresponding dialog box appears. The following sample dialog box would appear during the installation of Version 5.2.0, if (beta) version 5.2.0b1 would have already been installed on the target workstation.



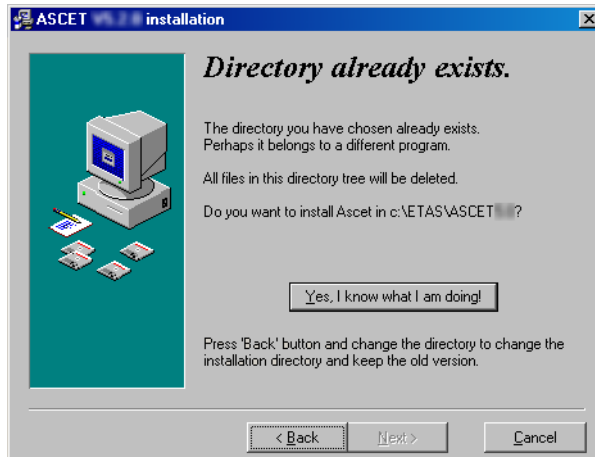
- Read the instructions carefully.
You are informed that an older installation has been found, and that files will be overwritten when you continue.
- To select a different directory, click the **Back** button.
- If you want to overwrite the existing files, click the **Yes, overwrite it** button.



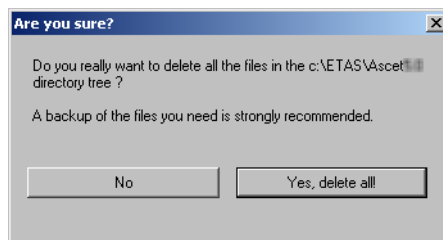
- Acknowledge the confirmation prompt by clicking **Yes, overwrite it!**
Click **No** to return to the previous window.

To overwrite existing directories:

If the directories specified during the installation already exist, and a complete ASCET installation does not exist on the target workstation, the following dialog box appears. This case occurs, e.g., if a previous installation process has been canceled.



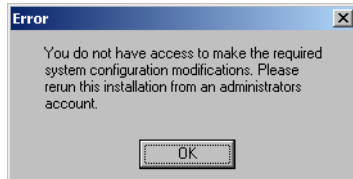
- Read the instructions carefully. You are informed that the directory you selected already exists.
- To select a different directory, click the **Back** button.
- If you want to overwrite the existing directories, click the **Yes, overwrite it** button.



- Acknowledge the confirmation prompt by clicking **Yes, delete all!**
ASCET is installed; existing files are deleted.
Click **No** to return to the previous window.

Performing an installation without administrator privileges:

This message box appears if the user name under which you logged on to Windows has no administrator privileges. After you acknowledge the message, the installation is terminated because administrator privileges are prerequisites for the installation of ASCET.



- Confirm by clicking **OK**.
This installation is terminated.
- Please contact your system administrator.
- Try the installation again after you have received the required privileges.

2.3 Network Installation

In addition to the installation from the CD, you can also install ASCET from a network drive on the PC.

The network installation provides the benefit that you can adjust data even before the actual installation on the workstation takes place (see section 2.3.2).

2.3.1 Providing Data in the Network

To provide data in the network, you have to copy the installation files from the CD to the desired network drive.

To provide data on the network server:

- Create a source directory on the desired network drive.

- Copy all data from the CD to the source directory.

Installation Log

The network installation of a user is logged in a file on the network. Therefore, *all* users need write access to the `x:\user` directory or to the directory specified in `install.ini` for the registration.

2.3.2 Customizing the Network Installation

You can customize ASCET (modifying specific default settings) even before the user installs ASCET on his workstation.

You have the following customizing options for the network installation:

- You can customize the installation dialogs (modifying the default settings for, e.g., directories, etc.).
- You can perform the ASCET installation fully automatically and without user intervention invisibly in the background.
- You can overwrite the files provided in the product data directory (default setting: `[drive]:\ETASdata\ASCET5.2\...`) with your customized files and/or add files to the existing directories.

Customizing Installation Dialogs

For the network installation in large enterprises, it is often necessary to customize certain default settings during the installation to fit internal standards and requirements. This is possible by means of the `install.ini` configuration file. This file is located in the installation directory.

The following example will show you how to modify the default settings.

To customize the configuration file:

- Open the `install.ini` file with a text editor.

The following is a typical example of an entry in this INI file:

```
;Sets the main directory of ASCET
;MainDir=c:\etas\ASCET5.2
```

- To modify the default setting, delete ";" (comment) on the line with the `MainDir` keyword.
- Change the path to, e.g.,
`H:\programs\etas\ASCET5.2.`

The entry should now look like this:

```
;Sets the main directory of ASCET  
MainDir=H:\programs\etas\ASCET5.2
```

- In the same way, modify all other entries in `install.ini` as desired.
- Save your changes and then close the editor.

When you now start the installation with `Ascet.exe`, the dialog boxes will show the new settings as defaults.

Note

The modified path settings affect the functions described below.

Automatic Installation

By calling `Ascet.exe /s` you can execute the ASCET installation fully automatically and transparently in the background, i.e., without any required user interaction. This will select the currently valid default settings. You can configure these settings in the `install.ini` file (see "Customizing Installation Dialogs" on page 30).

If you as the system administrator create a batch file containing the `ascet.exe /s` command, and configure the required settings in `install.ini`, the users can run the installation process themselves by executing this batch file without the need to enter any further information.

As this type of installation does not display any dialog boxes, you may want to provide some mechanism to inform the user when the installation has finished.

Customizing ASCET Files

The customization feature described below allows you to control the installation routine so that certain default files are overwritten with your customized files during the installation, or that other files are included in the installation.

In this way, you can integrate your customized databases, user profiles, and window templates in the installation routine.

The mechanism is relatively simple. Create a subdirectory named `InstData\...` in the installation directory and copy your customized files into it while maintaining the proper directory structures.

To create your customized files, install ASCET on a test computer, and use this to create the files.

After the default installation of ASCET, the `ETASData\ASCET5.2\...` directory contains various subdirectories with files that determine the ASCET default settings which you can customize. These include the following subdirectories:

- `Database\DB\`
The `db` subdirectory contains the default database. Here you can, e.g., create another demo database under `Database\DemoDB\`.
- `User\[user name]`, depending on the windows login
The `[user name]` subdirectory contains the default user profile. All configurable options are stored in this subdirectory.

Customizing Data for Network Installation:

- [Install ASCET on your PC.](#)
- [Start ASCET.](#)
- [Modify the user profile.](#)
- [Modify the database or add a new database.](#)
- [Exit ASCET.](#)

You have finished your customization and now want to integrate these files in the installation routine. You have two choices.

- Overwrite existing files having the same name with your customized files. To be able to do this, you must create a folder named `InstData\overwrite\` in the installation directory.
- Rename your customized files and add them to the existing files. There are no files with the same name that will be overwritten. To be able to do this, you must create a folder named `InstData\add-only\` in the installation directory.

To include your customized files in the installation routine, be sure to copy also the parent directories. Note that the `ETASData\ASCET5.2\` directory level must be the same as `InstData\overwrite\` or `InstData\add-only\`.

Examples:

```
InstData\overwrite\user\userDef.ini
InstData\add-only\database\additionalDB\
```


To integrate a modified user profile:

- Copy your customized file `ETASData\ASCET5.2\user\[user name]\Ascetsd.ini` into the `InstData\overwrite\user\` subdirectory.
- Rename the `Ascetsd.ini` file to `user-Def.ini`.
This ensures that this initialization file will be used after the installation for each new user.

To integrate a modified database:

- Copy your customized database, i.e., the `\database\DB\` subdirectory into the `InstData\add-only\...` subdirectory.
- Rename the `db` directory as desired, otherwise the database will not be copied.
Of course, you could also overwrite the `db` database by using the `InstData\overwrite\` directory.

When starting the installation routine with `ascet.exe`, the default files will be replaced with your customized files and/or your new files will be added in the corresponding directories.

2.3.3 Installing ASCET from the Network Drive

The installation from a network drive is described under "Installation" on page 17. You only need to know on which drive and in which source directory the installation files are located.

Note

To install ASCET from a network drive, you need write access to the log directory on the network drive (see chapter 2.3.1).

2.4 Uninstalling ASCET

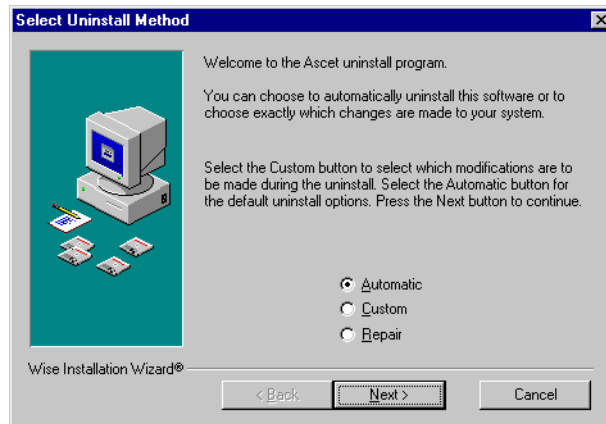
When ASCET is uninstalled, *all* add-ons on the computer are uninstalled, too. You can uninstall only the entire ASCET product family, not separate products as ASCET-MD, ASCET-SE or ASCET-RP.

2.4.1 Automatic Uninstall

To uninstall ASCET (automatically):

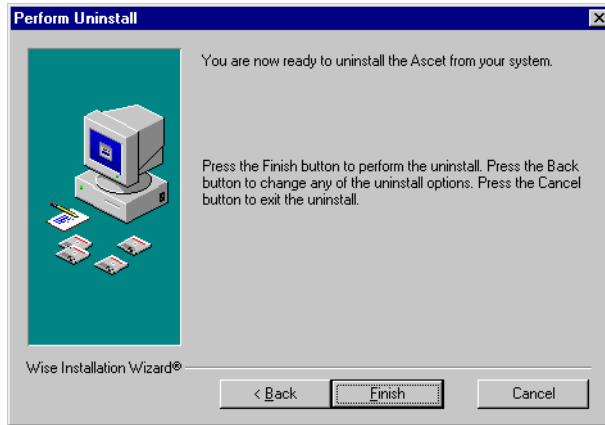
- From the program group of the start menu, select **ASCET Uninstall**.

The following window is displayed:

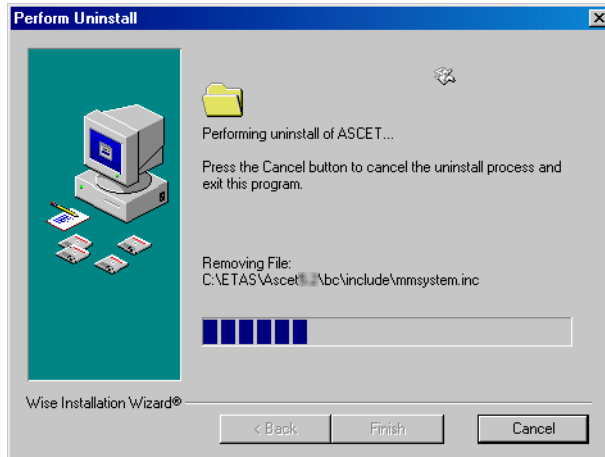


- Select **Automatic**.

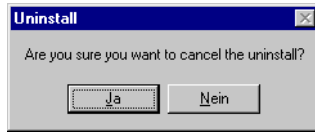
- Click the **Next** button.



- Click the **Finish** button to execute the uninstall process.



You can cancel the uninstall process. If you click the **Cancel** button, the following window appears:



Note

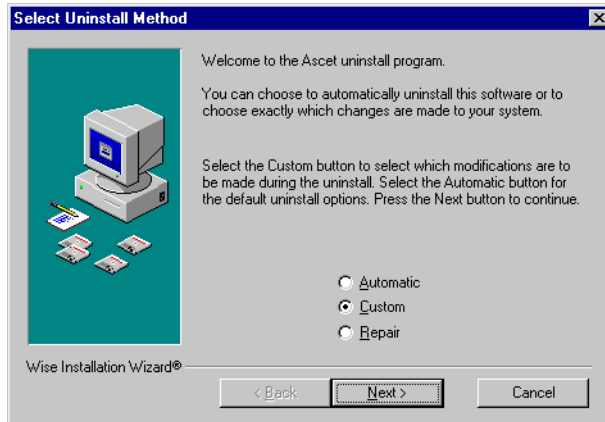
If data has already been deleted, ASCET must be reinstalled.

2.4.2 Custom Uninstall

To uninstall ASCET (user-defined):

- From the program group of the start menu, select **ASCET Uninstall**.

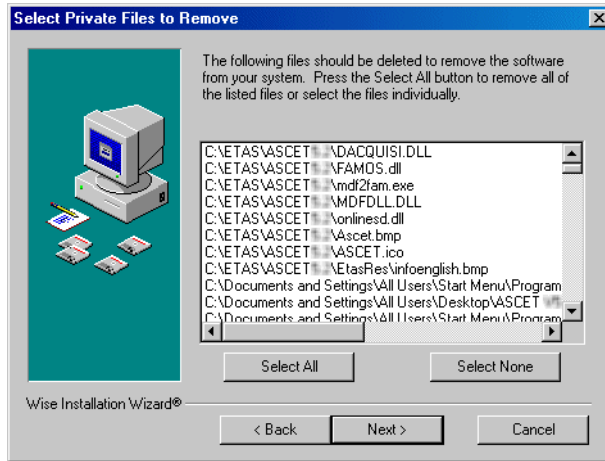
The following window is displayed:



- Select **Custom**.

- Click the **Next** button.

The "Select Private Files to Remove" window opens.



- In the "Select Private Files to Remove" window, select the files you want to remove.
- Click the **Next** button.
- In the "Select Directories to Remove" window, select the directories you want to remove.
- Click the **Next** button.
- In the "Select INI Files to Remove" window, select the * .ini files you want to remove.
- Click the **Next** button.
- In the "Select INI Items to Edit" window, select the * .ini entries you want to edit.
- Click the **Next** button.
- In the "Select Registry Keys to Remove" window, select the registry keys you want to remove.
- Click the **Next** button.
- In the "Select Registry Trees to Remove" window, select the registry folders you want to remove.

- Click the **Next** button.
- In the "Select Registry Keys to Edit" window, select the registry keys to be edited.
- Click the **Next** button.
- In the "Select Sub-Systems to Remove" window, select the sub-systems you want to remove.
- Click the **Next** button.
- In the "Perform Uninstall" window, click the **Finish** button.
- The deinstallation is performed.

You can cancel the custom uninstall process, as well as the automatic deinstallation, using the **Cancel** button.

Note _____

If data has already been deleted, ASCET must be reinstalled.

3 Licensing

Products and Add-Ons of the ASCET Product family are subject to license management. In order to work with an ASCET product, after the installation, you need a license file for your computer. Without this file, ASCET products can be installed, but you cannot use them.

3.1 Obtaining Licenses

The license file can be obtained from ETAS.

To obtain the license file:

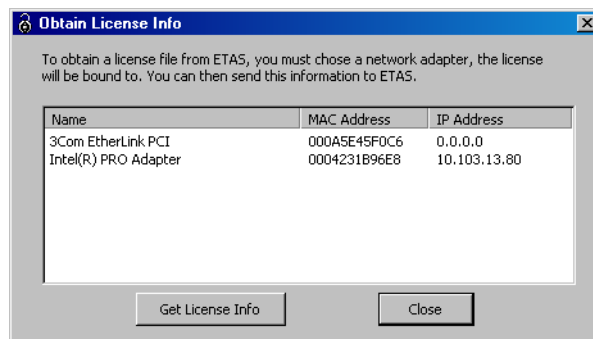
The license is tied to username and workstation. To obtain the information required for the creation of the license file, use the **LicenseInfo** tool in the ASCET program group.

Note

If you would like to request the license file before installing an ASCET product or add-on, you can start the program `LicenseInfo.exe` from the product CD-ROM.

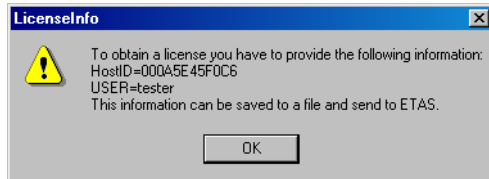
- In the Windows start menu, select the ASCET folder of the current ASCET version.
- In the ASCET folder, select **LicenseInfo**.

The "Obtain License Info" window opens. It shows the MAC and IP addresses for each network board of your computer.



- Select one of the listed Ethernet adapters and click on **Get License Info**.

The information about your computer required by ETAS to create a license file are collected and displayed in the "License Info" window.



- Click **OK** to store the information in a text file.
- In the file selection dialog window, enter path and name for the text file.
- Click on **Save**.

The file is created and opened in a text editor. Besides your host ID and user name, it contains input lines for your E-mail address, license number, and other specifications, as well as the ETAS contact addresses.

- Close the "Obtain License Info" window.
- Send the completed text file to ETAS.

You find the required license number on the license contract. You have to send a completed text file for each ASCET product you want to use, e.g. ASCET-MD or ASCET-MIP.

ETAS will send you the license file with the required license keys for your computer.

Note

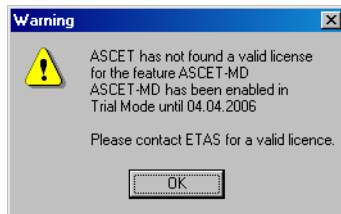
*The license file must **not** be edited. Otherwise, it becomes invalid.*

- Store the license file in the license directory intended for that purpose (see page 23).

By default this is the directory `ETAS\ETASShared3\Licenses`. It is possible, however, to modify the license file directory by modifying the environment variable `ETAS_LICENSE_FILE`.

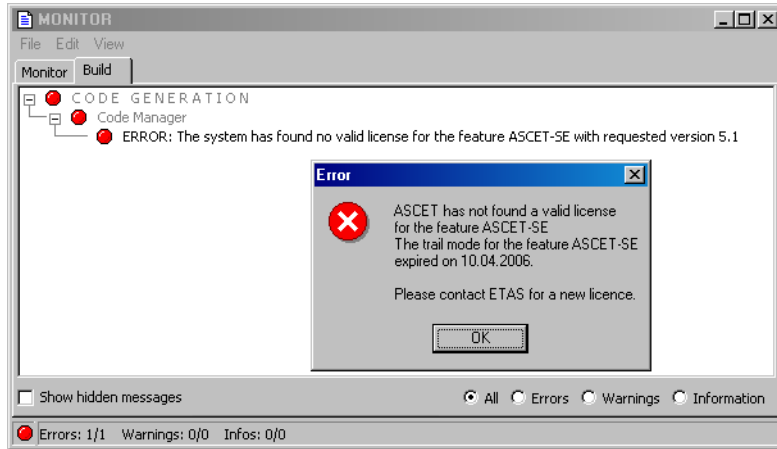
In this license directory, the license file is recognized automatically when you start the program.

If no valid license file is detected in the license directory specified during installation (see page 23), a *trial mode* is activated for ASCET-MD, ASCET-RP or ASCET-SE. That is a limited period of time during which the tool is fully operational, but a license file is searched regularly, and a warning is issued when the search fails.



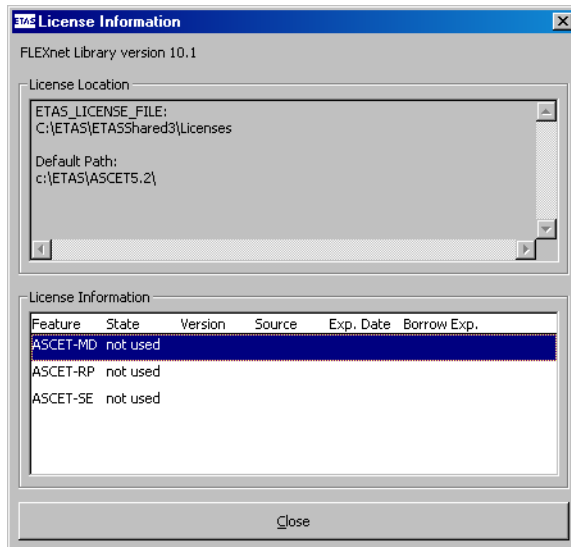
Add-ons such as ASCET-MIP or INTECRIO-ASC do *not* have a trial mode. Without a valid license file, they do not work.

Once the trial mode is expired, an error message is issued instead of the warning, and the tool does not work until a valid license is found.



3.2 Licensing Status

You can display the current licensing status at any time. The [Help](#) → [License Info](#) menu option opens the "License Information" window.



The upper part of this window contains information on the license location. The lower part contains a table with the current licensing status.

Column	Description
Feature	installed ASCET products or add-ons
State	licensing status; possible values are: not used – no licence for this feature has been requested licensed – licensed feature grace mode – feature in trial mode unlicensed – trial mode has expired
Version	version number of the feature as derived from the license file
Source	source of the license file; possible values are local license or a server name (or empty, if no license was found)
Exp. Date	expiration date of the license or the trial mode
Borrow Exp.	expiration date of borrowing (see chapter 3.3)

Tab. 3-1 Description of "License Information" table

3.3 Borrowing Licenses

When you are using a server license, but need a local license (because, e.g., you want to use a notebook in a car), you can *borrow* the server license for a limited time. (You do not need to borrow the license as long as you are connected to the server.)

To change the borrowing time:

The borrowing time is set to 60 days; you can change that value in the ASCET options window, "General Settings" tab.

- In the Component Manager, select **Tools** → **Help**.
The "Options" window opens.
- In the "Options" window, open the "Licensing" node.
- Enter the borrowing time in days in the "Borrow license for days" field.

- Close the "Options" window.
The next time you borrow a licence, the borrowing expiration date is derived from this value.
Existing borrowing expiration dates are not changed.

To borrow a license:

- In the Component Manager, select **Help** → **License Info** to open the "License Information" window.
You can borrow licenses for all features in not used state.

Feature	State	Version	Source
ASCET-MD	not used		
ASCET-RP	not used		
ASCET-SE	not used		

- In the "License Information" window, right-click the feature whose license you want to borrow, and select **Borrow license** from the context menu.

With that, you have borrowed the license. The expiration date is filled in the "License Information" table.

Feature	State	Version	Source	Exp. Date	Borrow Exp.
ASCET-MD	not used				
ASCET-RP	not used				23.03.2006

The full functionality of the selected feature is now available locally on your computer.

As long as you hold the borrowed license, it cannot be accessed by someone else. To avoid bottleneck situations, you can return the license before the borrowing time expires.

To return a borrowed license (normal case):

To return a borrowed licence, your PC must be connected to the network.

- Open the "License Information" window.
- Right-click the feature whose license you want to return, and select **Return earlier** from the context menu.

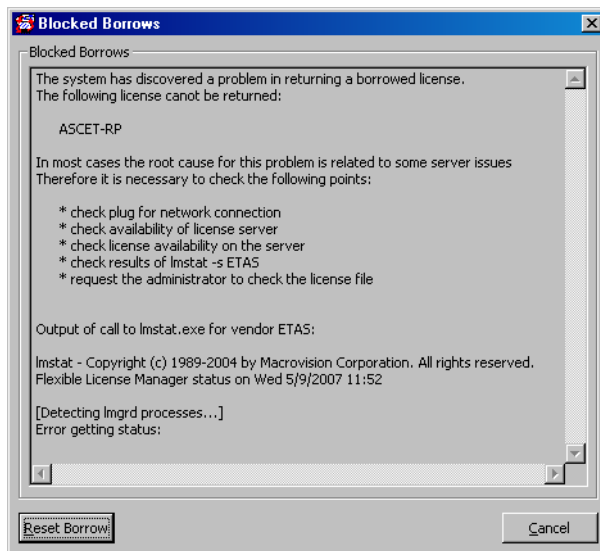
The license is again available on the server.

To return a borrowed license (error case):

If you borrow a server license, information regarding the borrowing is stored both on the server and locally on your PC. If the information on the server gets lost (this can happen, e.g., when the server was restarted without success, or with server errors), return via **Return earlier** does not work. You can use the command, however, to delete the local information from your PC.

- Open the "License Information" window.
- Right-click the feature whose license you want to return, and select **Return earlier** from the context menu.

The "Blocked Borrows" window opens. It contains a list of points you have to check.



- Check the following points:
 - Is your PC connected to the network?
 - Is the license server available?
 - Is the license on the server available?
 - What does the result of `linstat -s ETAS` look like?

- Ask your administrator to check the license file on the server.
- Click **Reset Borrow** only if the server information on your borrowed license is really lost.

Note

*When you delete the local license information while the server license information is still available, the license is **unavailable** until the regular end of the borrowing time.*

The information regarding your borrowed license stored locally on your PC is deleted.

4 Understanding ASCET

ASCET is a high quality, rapid development tool that ensures reuse of existing components. Its unique graphic development environment allows target-independent functional specifications. Control software for embedded systems can be generated automatically from diagrams. Target-identical prototyping provides early tests in the development cycle. The ETAS tool chain makes your investment safe and profitable.

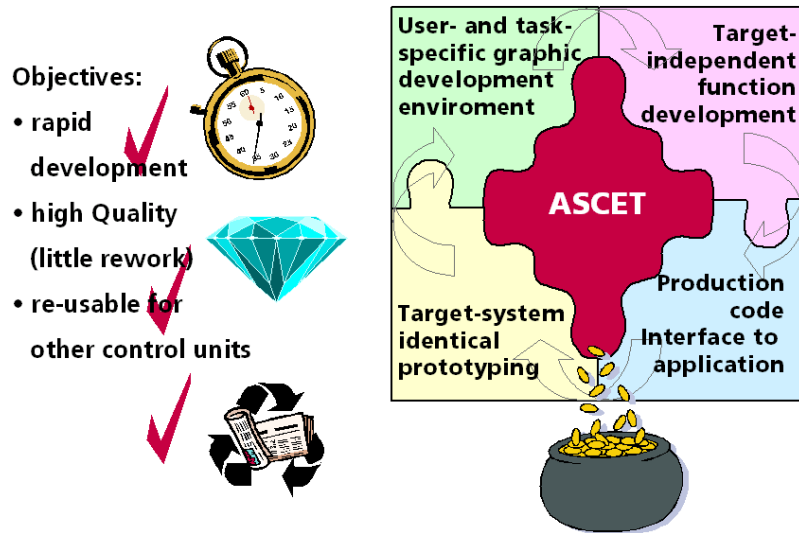


Fig. 4-1 Advantages of the ASCET development environment

4.1 Increasing Efficiency in Control Unit Development

For the past 20 years electronic control units have gradually made inroads in car production as well as other branches of industry. In the meantime there has been a drastic rise in the requirements for functionality, speed and networking capability. Technology and methods have changed enormously. Today, electronics are a key factor governing the success of new vehicle models. Development costs and the speed of development are gradually increasing in importance. The latest technologies are providing competitive advantages. ETAS is breaking new ground in this sector.

4.1.1 Modern Embedded Control Systems: Technical Mission

A characteristic feature of present-day control unit technology is the integration of many high-quality functions on a single processor chip. The microcontrollers actually used in practice are often equipped with several I/O and

communication channels. Since the outside world is embedded directly in the processor in this way, we normally speak about Embedded Systems and, when it comes to control technology, of Embedded Control.

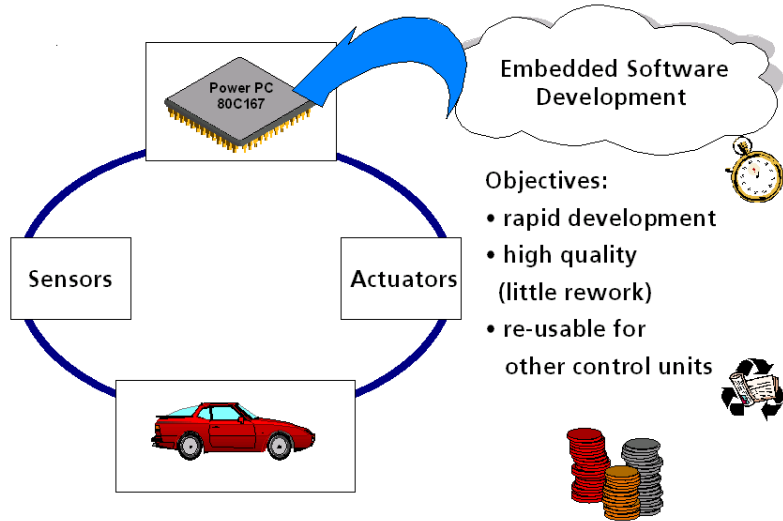


Fig. 4-2 Principle of Embedded Control System

Embedded Control Systems implement complex control circuits. Information on the current state of the controlled system is precisely recorded by sensors and processed together with previous data. The result is the output of control information via corresponding actuators. The coupling of a variety of system aspects is becoming ever more important. Only by networking the various control units can we master the entire system now, and implement present and future environmental requirements with respect to safety, emission cleanliness and fuel economy.

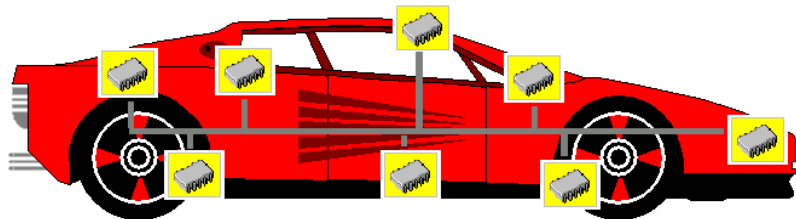


Fig. 4-3 The reality of Embedded Control

Vehicle electronics have already penetrated nearly all parts of the system. Historically, development started with engine and transmission control. But now the success story of electronic control units also includes the brakes and the suspension. Electronics have also had a visible impact inside the vehicle (airbag, cockpit, navigation systems, etc.).

Embedded Control Systems

Many details need to be considered when designing embedded control systems. Usually the control unit must incorporate numerous signals. In addition, the physical relationships between the signals are highly complex. If the design engineer wants to obtain fast results, he needs methods and tools that relieve him of additional difficulties when it comes to implementing the control unit on microprocessors. By abstracting the task and shifting it to the physical plane, it must be possible to reduce complexity to an acceptable measure and make the design "easy".

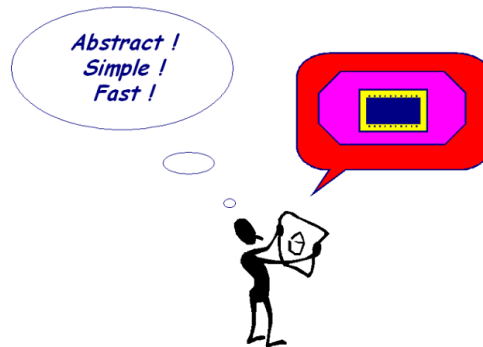


Fig. 4-4 Vision of Embedded Control

The vision of embedded control development systems is therefore based on requirements for abstraction, simplicity and speed. ASCET meets these requirements. Using ASCET, the design engineer can fully concentrate on the physics of the embedded control system, and design the control unit in an abstract and simple environment in order to finally generate an automatic code for the series launch. This achieves an exemplary speed during the design phase.

A call for abstraction, simplicity and speed have emerged from all sectors of software development over the past few years. The results have been visual methods for the analysis and design phases. The various techniques have now been combined in a universal language called UML (Universal Modeling Language). What are known as *patterns* are defined at this level. They translate the design engineer's experience into the most abstract shapes ever known. However, this approach still offers no support for describing real-time behavior and its transparency does not conform with our control approach. ASCET

offers all this: a reasonable amount of abstraction, clear support of real-time requirements and a visual description in the form of control block diagrams and state machines. And whenever possible, specifications can be transferred from UML to ASCET and back. The **openness** of ASCET opens up new perspectives here.

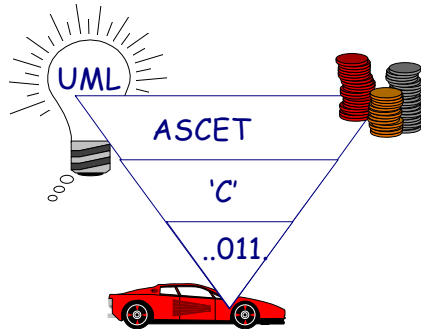


Fig. 4-5 Modern software development

With ASCET, the specification is translated into C code at the push of a button. It is then transferred to executable binary code using target-specific tools (compilers, linkers, debuggers) and loaded on the target. ASCET effectively shortens development time and therefore reduces costs. Moreover, existing sources (e.g. C code) can be reused, providing excellent support for migrating a pure C-development to ASCET. The abstraction of real-time requirements is fully met by our **OSEK**-compatible operating system ERCOS^{EK}.

But after executable code is generated, there is still a long way before development of embedded control systems is finished. During the application and test phases, a wide range of measurements are performed and parameters are optimized. This is where development systems must become application-oriented. The unique requirement for continuity throughout the development process is expressed by support for interfaces and format. ETAS meets these requirements totally with ASCET, INCA-PC and LabCar. A matching range of products with **standardized interfaces** offers the design engineer a high degree of convenience, permitting him to focus his attention on the main task

in hand - the development of embedded control systems. A parallel, continuous documentation is just as natural as the openness of the tools for integration in cross-project management (Configuration Management).

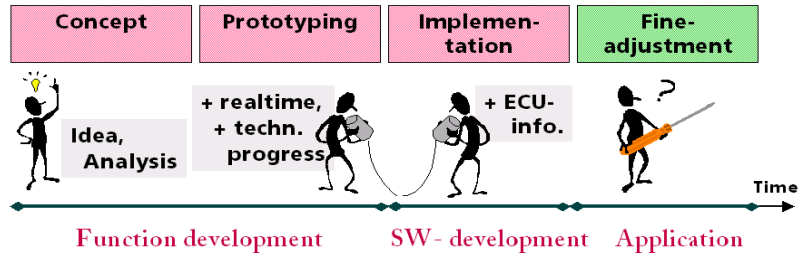


Fig. 4-6 Continuous process - open interfaces

No continuity within the process would mean time-consuming conversions of data. Misunderstandings, gaps in the specification or even errors would be the result. And these problems would crop up every time there is a change. On the bottom line, this results in enormous costs which could be avoided by selecting a continuous tool chain from the start. ASCET is continuous and open. Using ASCET saves costs.

Focus on the Car Industry

ASCET was mainly developed for use in the car industry. The result is a number of requirements which were previously undiscovered by other branches of industry. On the other hand, many general concepts which can be of use to other development tasks have flown into our tool chain.

A characteristic feature of automobile embedded control systems is requirements for simple control concepts such as maps. This is where the constraints of memory size and performance (computer time) become very noticeable. Only through simplified models of ideal controllers can we achieve the excellent results desired in monitoring and controlling physical processes. Add to this a large number of requirements which occur in other application areas, but which are unique in a network: quality at this point is not a universal factor. But when considered alone by the volume of the end product produced, it becomes a critical factor for embedded control systems. Another factor is the safety aspect. Many control units implemented in the motor vehicle affect safety-relevant or even safety-critical operations. Brakes, engine control and even window lifts (danger of jamming a limb) are characteristic examples. The topic of networking has already been mentioned in this context.

But the hardware used in development must also meet special requirements. Simulation systems, measuring and adjustment systems as well as test systems must be up to the aggressive conditions of summer and winter tests. They must have excellent electromagnetic compatibility and, of course, satisfy the highest performance demands.

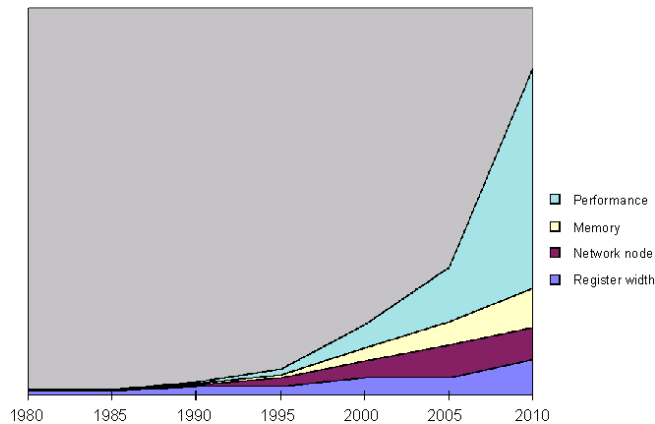


Fig. 4-7 Requirements in the car industry

The innovation cycles in the car industry are mainly determined by trials. The products themselves have a much longer life cycle and they are also face-lifted in this period. Enhancements in microprocessor technology are therefore viewed from a long-term aspect or must be adaptable without affecting development activities (e.g. more advanced memory technology). The starting point is therefore a relatively approximate model in which such things as register width, i.e. the measure for processor performance, doubles only every five years. It must be taken into account that large overlaps take place, i.e. when the first 32-bit processors were introduced, there were many projects running on 16-bit technology and even some that were still running on 8-bit technology. The same applies to the upcoming innovation step from pure integer arithmetic to floating point.

4.1.2 Development Processes: Economic Challenge

Development processes are always related to a particular company. However, there are many general requirements which are ultimately reflected in the supporting tools.

In all modern developments, the strategic foursome of time, costs, quality and flexibility play a decisive role. Development processes must be measured by these factors. The continuous tool chain is a vital step in this direction. But success only happens if every member in the chain makes its full contribution. ASCET and the other ETAS tools are more than up to this challenge. They help

to save time through their standardized interfaces and their unique alignment to the technical requirements of embedded control systems. This is also a major contribution to reducing costs. There are also cost savings in the well-arranged control-related modeling features. Together with early prototyping, this helps to avoid superfluous recursions and failed developments. It also improves the quality of development results. Quality is enhanced by the automatic code generation feature in ASCET; this closes the usual implementation gap. The specification and the product have a 100% match. Laborious rework becomes a matter of the past. Together with the unique ASCET database design, development work becomes flexible. Modules can be linked in any combination and the automatic code generation feature performs the necessary optimizations and adaptations. Finally, ASCET offers easy-to-modify interfaces. They provide flexible support for practically any development process.

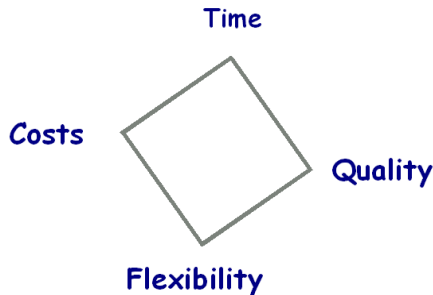


Fig. 4-8 The strategic foursome—product development requirements

The special economic challenge in the car industry is handling large volumes. This mainly has consequences for production processes. From a development engineer's viewpoint, there are other aspects, e.g. cooperation between several suppliers on a project. The tool chain is therefore subjected to requirements which can be grouped under the term of team support. It must be possible to manage data jointly and exchange data reliably and reproducibly over large distances. And the topic of know-how protection must not be neglected. ASCET offers the right concepts for this and has the necessary interfaces for the tools.

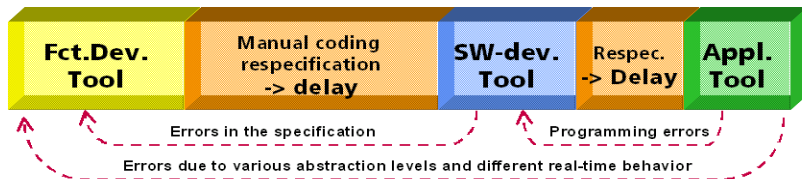


Fig. 4-9 Conventional development process

In hotly contested markets, the pressure of costs and innovation is equally permanent. The aim here is to achieve synergy beyond projects. This can only be achieved with cross-modular solutions (identical parts concept). When it comes to developing embedded control systems, this means introducing standardized interfaces at any level (hardware, operating system, protocols, functions) as far as possible. This solution achieves the necessary shortening of development cycles. Standards are common practice at ETAS: Our tools support or set the standards in the field of embedded control systems (e.g. ASAM, ASAP, CAN, MSR, NEXUS, OSEK, VME). We are actively engaged in upcoming standardization requirements.



Fig. 4-10 Optimized development process with ASCET

Present economic requirements can only be met if development is both efficient and effective. Doing the right thing is the guideline, both in small or large companies. Everyday working with ASCET brings more efficiency in the development of modern embedded control systems. The simplifications in everyday development work described above liberates capacity that is necessary so that the engineer can concentrate on the job she is supposed to do— **inventing and implementing new, innovative control units.**

4.1.3 Innovative Technologies - Technological Visions

Software for Embedded Control Systems has been produced for several decades. Over the years there have been many changes to the programming languages and tools used. But the basic methodology has remained unchanged. Only microcontroller experts have been able so far to carry out the complex programming of these systems. The chances of using a debugger for troubleshooting were and still remain very problematic. This is only possible to a limited extent (real-time problem). Function developers have no chance of obtaining results directly. In the meantime, ETAS has revolutionized this sector: The leading technology of ASCET has made it possible for the first time to verify a draft control design directly on the target system.

Future orientation and innovative strength are part of our character. They are the driving forces of progress and improvements. Our products correspond to this paradigm. We therefore set the standards. And we align ourselves fully to the wishes of our customers when optimizing our products. Why don't you develop with us the future of embedded control systems? This will make sure that progress will not overtake you.

Code Generation

The era of bit coding is past. Assembler is also a rare choice for a serious implementation language for embedded control systems. Today many people are developing in C. This high-level language marks the end point of a development which has always been centered on the target system. But this cannot implement any cross-platform control concept. Reusing physically identical data has thus become a horror scenario. You start the same thing from the beginning every time. Truly a Sisyphean task.

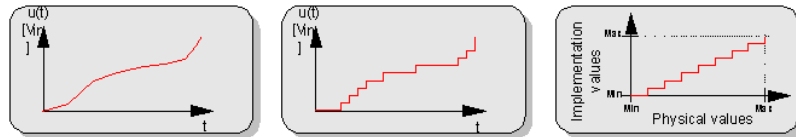


Fig. 4-11 From physical parameters to implementations

In this situation, ASCET offers something incomparable. For the first time it is possible to move directly and automatically from the graphical control block diagram to software on the microcontroller. The ASCET code generator is unique, efficient and simple to use. The task given to the design developers at ETAS was anything but trivial. The difficulty first lies in the correct abstraction of the requirements: Microcontrollers still work with integer arithmetic. They have limited memory space and a wide variety of hardware architectures. Through the availability of ASCET-SE for various microcontroller targets, we have succeeded in encapsulating these differences and obtaining a standardized interface for the user. The result: There is no longer any problem in changing from one microcontroller to that of another vendor or even to the latest

upgrade. Reuse is no longer simply a buzzword but has become practical reality in the designer's everyday work. And to keep it this way, we convert all the main new processor developments into the corresponding ASCET-SE ports.

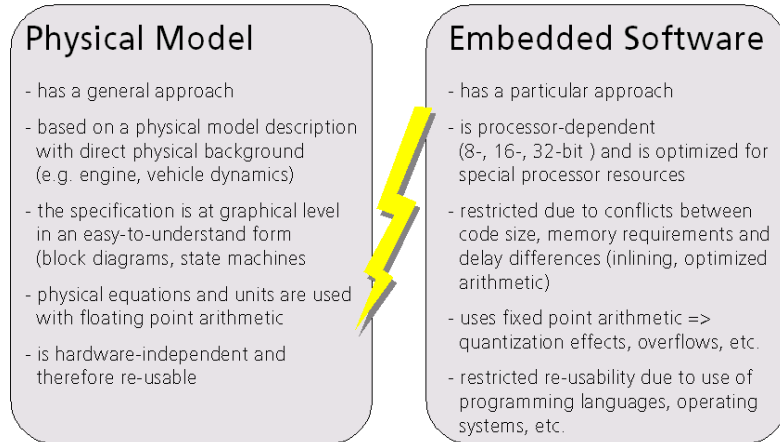


Fig. 4-12 Contradictory requirements of physical modeling and embedded software implementation

ASCET-SE generates C code. It uses the usual tools for microcontroller programming (compilers, linkers, locators and debuggers). But there are new challenges here, in particular in the configuration on handling these tools. Here, too, ASCET has set milestones. But progress does not end at this point. For example, innovative technologies link the topic of debugging with the application of electronic control systems (measurement and adjustment). This achieves complete control over real-time response on target systems. For the first time you can work directly on the target system at physical control level. The benefits are obvious: There is no more need for permanent thought transfer from control level to processor bit level.

Prototyping

In the meantime simulation technology has become established as a fixed component of complex control system development. It is a way of consolidating new functions at an early stage. In most cases, simulations are based on offline time steps. The influence of real-time requirements on the control system is not taken into account yet. Accordingly, sensors, actuators and the physical systems on which they are based, what are known as process models, are simulated for the simulation. It is therefore a purely software solution. This is the reason for the term "software in the loop" (SIL). But it can only solve the first step in the development of innovative control systems, i.e. the concept itself. However, event-oriented online simulation requires sensors, actuators

and the real process running in real-time. The operating system is then integrated at this point for the first time. The first software prototypes therefore arise from the simulation using hardware in the loop (HIL). The benefit here lies in the more realistic simulation of control requirements. New control strategies can be tested and adapted in the target environment under realistic conditions. A distinction can be made between two cases here: At an early prototyping phase, the control algorithm is still treated on the basis of physical parameters. Quantizations and overruns are largely neglected. They need only be considered at system boundaries (sensors and actuators). The next phase involves the implementation aspects. Finally it is only a small step from prototype to the product, which only consists in changing the computer hardware. The path from the idea to the product is thus a step-by-step method. At any time the designer has full control over the complex overall system and need only concentrate his work on one aspect of the overall task.

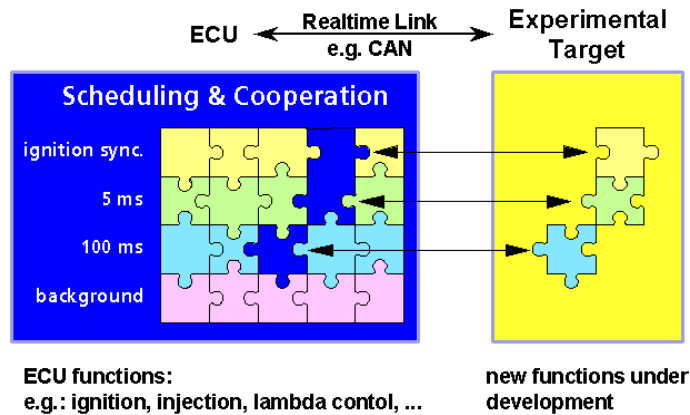


Fig. 4-13 Bypass technology - the basic idea

Prototyping can also be performed using an existing control unit. A distinction must be made here between the situation in which only part of the total control system is simulated and the rest runs on a series control unit (bypass) and the case where only the signal conditioning system of a series control unit is used (fullpass). There is also the possibility of supplying an existing control unit (but different to the target system) with a completely new program (development control unit). All these methods have one thing in common: the target system uses identical prototyping. As early as the prototyping phase, functional response can be studied in the necessary accuracy. The impact of quantization, overflows and real-time effects can thus be mastered early on. One condition for using this method is the equivalence of the simulation with the software version on the target system, i.e. the series control unit. ASCET offers

this special condition. In ASCET, online simulation as well as the series code are based on our real-time operating system, ERCOS^{EK}. Even quantization is treated in the same way in the online simulation as in the series control unit.

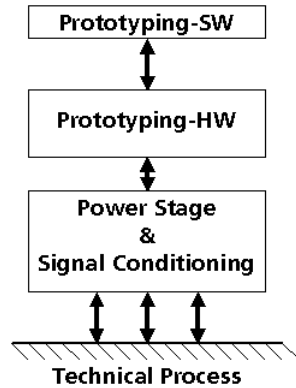


Fig. 4-14 The principle of prototyping

Another important aspect in prototyping is how to handle the environment. As already mentioned previously, it is useful to simulate the physical systems on which the sensors and actuators are based. These are time-continuous systems as opposed to the discrete modeling of control unit functions. They are normally presented by differential equation systems. However, only very simple solution algorithms are available for real-time handling, since a step-width control is not really suitable for this type of application. ASCET offers these methods in a single tool, i.e. with identical interfaces and workflows, from a single source, so to speak. Process models can therefore be integrated in the control development homogeneously if, for example, not all the physical components are available for prototyping at the same time. In the extreme, an entire vehicle can be integrated in online simulation (LabCar). Integration in continuous systems in ASCET brings benefits in day-to-day work. Providing and updating special interfaces and the necessary change in mindset are no longer necessary in a different tool environment.

4.2 Continuous Support for Embedded Control Systems

A major factor governing the increase of efficiency is the continuity of the tool chain. Any rupture in this environment will inevitably lead to faults in the development flow, cost, intensive recursions or to a manual rework. This can be avoided by using the right interfaces and formats. This situation demands standards in the form of migrations to new systems, i.e. the interfaces and formats must be supported by as many tool manufacturers as possible. Here, ETAS is actively participating in setting and propagating standards, and offers the necessary interfaces and formats in all the tools it produces. However, ETAS is the

only vendor who can supply the customer with the complete tool chain from a single source. Continuity for us, therefore, is not only a buzzword, it is a tried and tested code of practice.

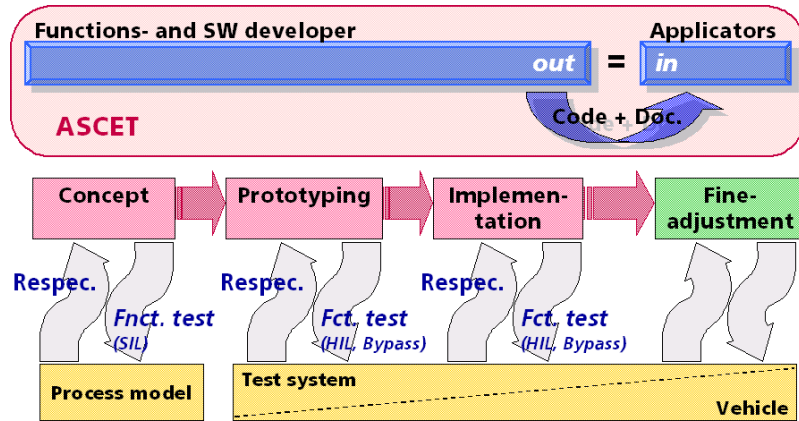


Fig. 4-15 Continuous development process

4.2.1 Entry-level Technology Bypass

New technologies and algorithms are introduced efficiently by the unique, executable specification of functions in the bypass. Since the majority of control functions can be taken from a particular series version, there is always a fixed status on which to build. This procedure is therefore an excellent way of introducing innovative modeling and simulation technologies such as ASCET. On the other hand, ASCET is also prepared for this situation. The Add-On ASCET-RP provides a work environment which makes entry easy. It also provides results that are quick to implement and provide useful information.

Before a bypass can be integrated, a modification of a particular function must be made in the control unit, i.e. in the series software or the near-series software. This must be provided by the software producer. The modification can redirect the normal function call from the function request to the bypass computer. A software switch is set for this purpose. If the bypass fails to reply to this request quickly enough (safety), an emergency program is switched and the values are used from the series function running in parallel. This type of modification is frequently provided for cooperation between suppliers and manufacturers.

Sometimes it may be useful to access new or other data of the existing series software. All that is needed for this are the addresses and the control unit which are usually provided for calibration purposes in the form of a control unit description file (ASAM-MCD-2MC). It is also possible to measure and

adjust data using a calibration system in the series control unit at the same time. This may become necessary as a result of interaction between the new function and the old state.

ASCET has the necessary conditions for integrating series states, interfaces to the control unit and the interfaces to actuators and sensors that are also frequently required. We have tested and proven the joint use of ASCET with our INCA calibration system.

4.2.2 Prototyping

A second area of application of ASCET is the field of prototyping. The prime task here is to develop new ideas from the start. In many cases, there is not even a series state to fall back on. Prototyping covers everything from the simple functional physical simulation, the study of quantization effects and real-time estimates through to the complete transfer to a control unit. At any level, it may be important or even necessary to fully implicate the control unit environment in the form of sensors and actuators. This occurs either in the form of a process simulation or by including real hardware in a closed control loop. It ensures the feasibility of a control concept and reveals more information about the exact requirements.

ASCET provides the necessary computer performance for these applications in the form of universal processor boards and supplies the external interfaces based on our ES1000 VME bus system and plug-in modules.

But the work does not stop here. It is possible to equip small fleets with these prototypes and simply present cost-effective solutions for the entire development process through to pilot run maturity by simply opening our simulation environment for application systems, e.g. INCA-PC. Our unique code generation technology goes one step further: You can achieve series maturity very fast by porting a few implementation data to a microprocessor target system and the automatic code generator. If this has to be tested in advance under aggressive conditions, the ETAS ES400 development control unit with ASCET can be used.

4.2.3 Automatic Code Generation

Mapping a functional model on executable program code is the main challenge in developing embedded control systems. The objectives at the physical modeling level and the control unit level could not be more divergent. Physical modeling is expected to be graphical, hardware-independent, reusable and to support physical data types (floating point arithmetic). It should also be easy to understand through the use of block diagrams and state machines so that designers can use the documentation as a reference during the application phase. But at the control unit level, i.e. the embedded software, implementation is a totally different animal. What is expected here is a code specially opti-

mized to the existing microprocessor with the appropriate memory devices. The code size and runtime are the prime parameters. The data types are normally integer (fixed point arithmetic).

If you want to avoid laborious, error-prone manual coding, the only way to link the two worlds is by devising automatic code generation. This is based on splitting the specification into a physical modeling part and an implementation information part. ASCET offers a step-by-step migration from pure C implementations to the new physical description methods. The C code generated in this system can also be used in other developer projects.

This approach continues through the experimental environment, e.g. quantization effects or runtime problems can be examined at an early stage. Since both the control unit program and the experimental environment are based on the same mechanism, we prefer to speak here about executing specification on different target systems than about simulation. To pursue this thought, we arrive at the question whether it is possible to run the application on the experimental system. When it comes to ETAS tools (hardware and software), we can answer this question with a clear Yes. Development is therefore continuous from start to finish.

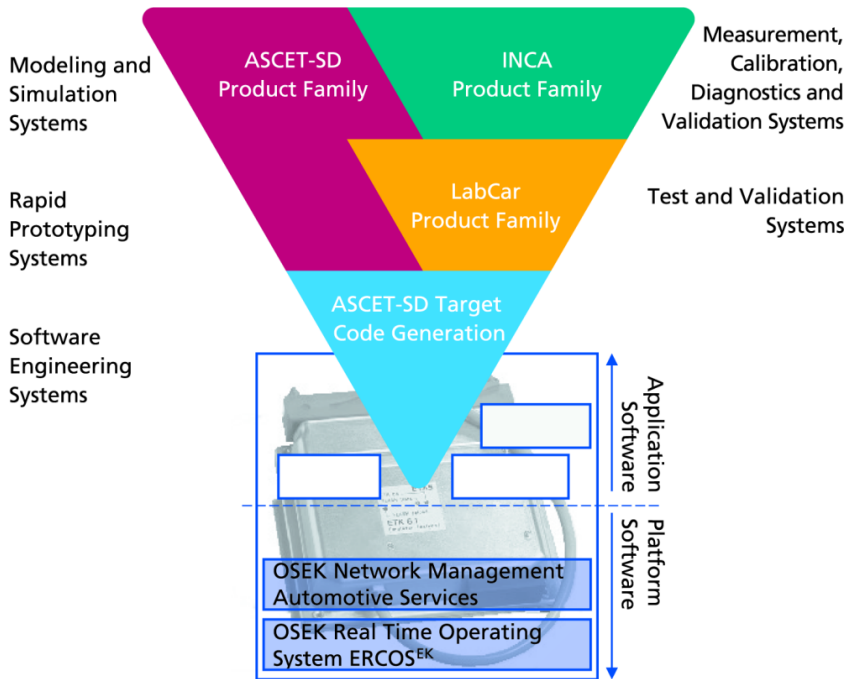


Fig. 4-16 Using tools in the embedded control system environment

4.2.4 Other Application Options for ETAS Development Tools

The unique ASCET code generator for series control units can be used in two different ways, i.e. as an additional programmer or as an integration tool. In the first case, it is very easy to imagine it as an entry scenario to the new technology, in particular for major projects. You can apply ASCET as an integration tool in all projects with good results. But even in smaller projects, an entry-level user can also obtain excellent results very quickly.

ASCET as Additional Programmer

When ASCET is used in this application scenario, it replaces a conventional workplace (specification and manual coding). The results of these activities flow as before into the total development in the form of C code modules. At the same time existing code can be converted in ASCET step-by-step (re-engineering). The result is executable code.

Transferring the remaining infrastructure, e.g. generating the control unit description file (ASAM-MCD-2MC) for the application, managing data stocks, etc., must be done by hand as before. On the other hand, ASCET generates the necessary information for this.

Team cooperation is fully supported by ASCET. There is an interface for linking configuration management (CM) tools. It is useful for the common management of ASCET specifications. High-performance import/export functions help you work without CM tools.

ASCET as Integration Tool

Complete development support on ASCET has the advantage that all components are optimally harmonized. There is no additional effort needed for converting between various tools or for integrating different sources. A single environment will produce specifications, implementations and test cases for a control unit. It also standardizes the management of different information sources.

Integrating existing ASCET modules mainly involves configuring the operating system. A definition is made regarding the form for invoking each of the functions and how to exchange this information. A distinction is made between cyclical tasks processed in a periodically recurring time pattern, and event tasks which start when a particular event occurs (interrupt). The related parameters can be simply set in ASCET. In addition, you can generate monitoring information at this point in order to perform further analyses of the control algorithms (e.g. run-time consistency).

Operating System and Components

Planning the various tasks (scheduling) is based on priorities which the design engineer stipulates. The ETAS operating system ERCOS^{EK} supports both cooperative and preemptive scheduling. The main task of the operating system is to transfer consistent data between processes. This takes place by using messages. When an interrupt occurs, the context of the current task is saved and later restored after exception handling.

Drivers for HW components only belong partly to the operating system since microprocessor peripherals can differ drastically depending on the application. What is known as hardware encapsulation accesses partly the hardware directly and partly the operating system. The abstraction of the functionality in the form of a data interface can be kept relatively general and categorized according to peripheral types (e.g. AD/DA converter, PWM, CAN). Above the operating system and HW encapsulation plane, there are protocols for application and measurement, diagnostics and bypass support. They are grouped as automotive services. Together with HW encapsulation and the operating system, they form the basis for the control unit application software. This modular model for control unit software has a number of benefits: Elements can be easily tested and exchanged. In many cases, different developer groups actually work on these parts. This division of labor supports rapid porting of an executable state to a new control unit.

In addition to the ERCOS^{EK} operating system, ETAS offers suitable HW encapsulation and automotive services. Based on this standard core, entry becomes much easier to the development of control algorithms since there is little need to bother about lower software planes.

4.2.5 Interfaces and Standards in the Tool Chain

Standards have a major significance worldwide in industry. Global operations without standards is inconceivable nowadays. Standards secure communications both at functional and technical level as well as at human level. Standards are an active contribution to protecting investment. Standards also offer the flexibility of exchanging development results or tools from different origins. They offer mechanisms to expand existing information easily and at low cost.

ETAS has made standards into a core concern. Our mission is to set and support standards. We do this by participating in standardization bodies and in the development of our products. In this way, many significant milestones have already been reached.

Our rapid prototyping hardware is based on the VME industry standard for buses. Our systems are therefore easily expandable by just adding new plug-in cards. Control unit interfaces have also been standardized in the car industry. The ASAM-MCD standard defines the link between the control unit and the

application system. The hardware is addressed via ASAM-MCD-1b drivers in the application or development system. The ASAM-MCD-2MC description file contains the necessary information about the addresses and conversion equations in the control unit software. ASAM-MCD-3MC defines unique interfaces to test stands.

On the topics of documentation and data exchange at the development tool level, an automobile standard has emerged from the MSR consortium. This ensures consistent development at all times at different locations by different partners using different tools.

At operating system level, the OSEK standard is becoming widespread for vehicle control units. ETAS is making an active contribution here. Work is continued both on the operating system kernel as well as on interfaces for communication and networks, i.e. the topics of HW encapsulation and automotive services.

On request, ETAS can also offer interfaces to other tools which do not yet rank as standards but which support the corresponding targets, e.g. data or model exchange with other development tools (e.g. MATLAB Integration Package). This simplifies the change-over to ASCET without losing expensive and costly work results.

4.3 ASCET Development Environment in Practise

The ASCET development environment for electronic control units is the solution for all the requirements discussed above. The innovative technology of ASCET is successful in improving the factors of time, costs, quality and flexibility in the strategic foursome. This is supported effectively by ASCET's structure. The basic ASCET package contains a number of specification options. Block diagrams, state machines, text specifications and C interfaces provide the design engineer with the right description options for control algorithms. Even the operating system configuration is graphical and can therefore be performed quickly and simply. The control process can be modeled and included

in the development. A continuous database supports cross-project reuse. The following sections concentrate on the technical characteristics of the ASCET development environment.

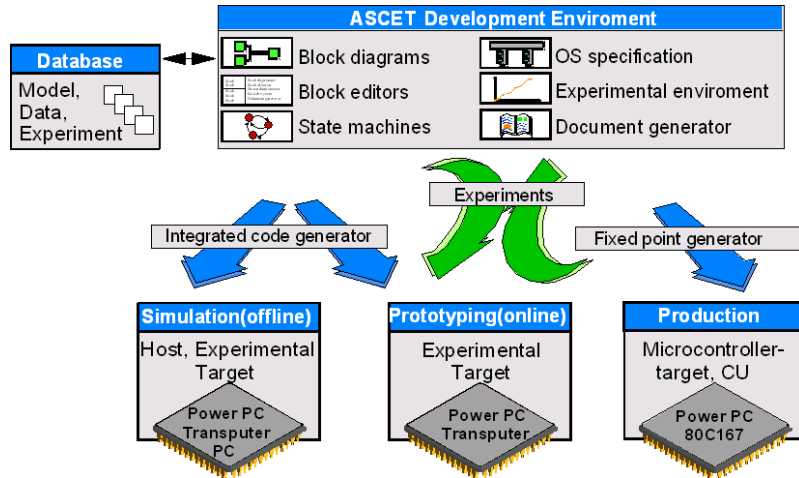


Fig. 4-17 Structure of the ASCET development environment

Below the specification level comes code generation. It has variants for different target systems and is based on the user's implementation information. The separation of physics and implementation makes the direct reuse of control units possible. Code generation supports the following scenarios:

- Physics experiment
- Quantization experiment
- Implementation experiment
- Controller implementation

Code generation affects both arithmetic and memory handling as well as the tasks and processes of the operating system. Platform dependencies and project-specific modifications are encapsulated in a practical way. This is more than just a pure code generation function, it is a general integration package which contains all the characteristics for interfacing any target system.

A high-performance experimental environment permits direct access to all data in the executable specification and the control unit program. Even during the real-time execution of the program, data can be manipulated graphically and displayed simultaneously. The user therefore has full control over the program.

Well thought-out hardware and software systems offer enormous flexibility in prototyping work with ASCET. Existing sensors and actuators can be integrated in a closed control loop, thus permitting a step-by-step development from the prototype to the product.

Bypass techniques support this process and offer a cost-effective entry to this innovative technology. The system provides continuous support for most conventional hardware interfaces (ETK and CAN).

Another advantage for our customers are open interfaces. Data reuse and investment protection have thus become practical reality.

4.3.1 Physical Specification of Control Systems

The specification of control algorithms must be based on the design engineer's viewpoint. Use is made of tried and tested graphical methods for block diagrams and state machines. However, in most cases it is much easier to formulate a mathematical expression directly than to create a block diagram. ASCET therefore supports text specification in JAVA-conform syntax. This section explains this method in more detail. It will take a closer look at the operating system configuration and control process modeling.

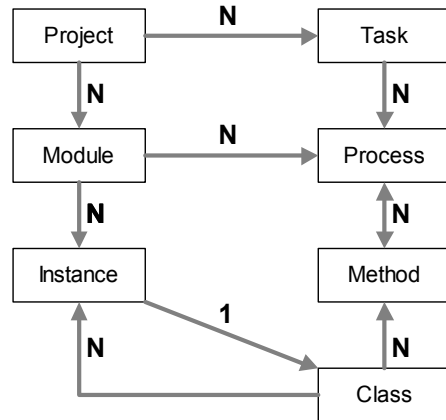


Fig. 4-18 Relationship between ASCET specification elements

The specification in ASCET consists of a number of dependent elements. The executable specification or the operating system configuration is continuously managed in a project (partly for implicit simplification, i.e. with a non-visible default project). Projects consist of a number of modules and tasks. These elements again contain several processes which are specified in the modules and scheduled for execution in the tasks. Modules encapsulate several class instances as objects. Nevertheless, modules only occur once. Each instance has precisely one class which contains the related methods. As opposed to a mod-

ule, a class can have several instances. Methods differ from processes in that they contain arguments and return values. Inter-process communication is provided by messages.

Both classes and modules can be implemented by block diagrams or text specifications. C code modules and classes are also offered. State machines are treated as classes.

Classes are regarded as the main representatives of reusable elements. Accordingly, ASCET offers a large library of useful classes from the field of control technology as well as general software engineering in the database.

All the ASCET elements mentioned above contain implementation information. In this way, they can handle target-specific or project-specific variants.

Block Diagrams

The block description of control systems is based on interfaces in the control process. In ASCET, this approach is supplemented by the element of encapsulation which is known from object-oriented programming languages. ASCET blocks represent objects which encapsulate items of information and are interconnected by interfaces. Object-based presentation ensures that they can be re-used easily and reliably. The block diagrams contain a purely physical presentation of control algorithms. Implementation information is easy to add to the elements using the block diagram by means of editors.

In traditional block diagrams, the precise order of computation is not always defined precisely. In addition to typical data flow elements (e.g. variables, characteristics, arithmetic operators, etc.) block diagrams in ASCET contain control flow elements such as branches. The control flow is shown in a well-arranged presentation by separate line and link types. In ASCET, you can also specify the processing order of block operations directly by assigning block attributes to the graphic. Complex algorithms can at last be presented precisely in graphic form by means of sequencing and control flow elements. These are then very easy to process.

An application-related view concept has been devised to support know-how protection for cross-company work. The user defines in a number of different views what blocks can be displayed or concealed in the documentation. ASCET also lets you define password-protected access rights at class, module and project levels.

State Machines

Hierarchical state machines are in widespread use in control systems. This modeling method is of special interest for systems where different control strategies are required depending on the working point. Trigger conditions and state methods can also be specified by block diagrams or even by texts. The user can select the best way of presentation.

Text Specification in ESDL

Contrary to the development of C code, the text specification of control algorithms in ESDL (Embedded Software Description Language) is a portable physical description. Instances are also encapsulated in classes and modules which can be enriched with target-specific implementation information. ESDL is based on JAVA since this C-like language is widespread and is easily learnable. Specifications in ESDL, state machines and block diagrams can be mixed in any combination, i.e. a class may contain some instances in block diagram form or others as text specifications in ESDL. As opposed to C, ESDL requires no pointer arithmetic since all the objects can be directly addressed. There is no dynamic instantiation. In other words, in ASCET, all objects are fixed at compilation time. This allows an early consistency check of the control unit programs and shows that the available memory is indeed sufficient. This also applies to handling arrays, matrices, characteristics and maps. These objects have a fixed access protocol as classes and require no pointer arithmetic.

Integrating C

C code within the context of ASCET must be regarded at the implementation level. Inevitably, it is target-specific and is also managed as such.

C sources can be integrated in two ways, i.e. as internal or external C code. With internal C code, the sources are managed in the ASCET database in the same way as ESDL classes. External C code, on the other hand, is stored in the user's final system and can therefore be used directly for other applications (see above: "ASCET as Additional Programmer"). Binary code can also be integrated if C-sources are not available.

The use of C code blocks is useful if the project includes special target-specific drivers. The interface is bidirectional. Methods can be invoked from any class by C code blocks. In addition, the C code also has an access option to other physically specified objects. It is obvious that the interface deserves special attention since the details of implementation are vital.

Operating System Configuration

The operating system interface is located in the so-called project (see above). Priorities are assigned for tasks incorporated in the scheduling. In addition, each task has other attributes which are vital for scheduling the processing order, e.g. whether they are cooperative or preemptive, whether they are cyclical or whether they are started at an external event or only initially. The operating system is then configured on this basis. Other information is taken into consideration. The processes invoked in the tasks communicate by means of

messages. Messages from various modules are linked by identical names. In the code generator, this information is checked for consistency (use of messages at several points, handling global variables, etc.).

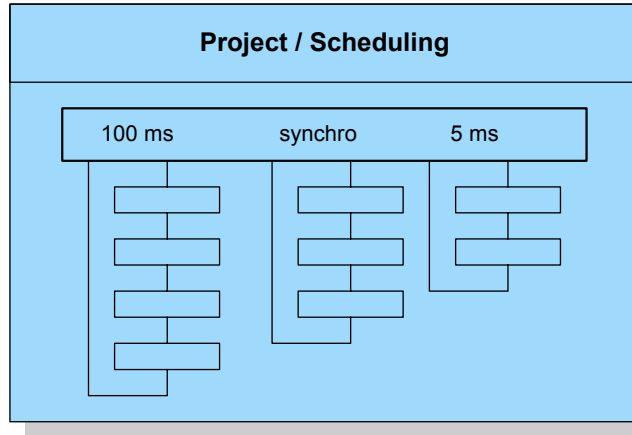


Fig. 4-19 Processes and tasks within the project

The operating system is configured visually in a simple editor. This provides the design engineer with a continuous overview over the total system. Changes can be made to scheduling very easily and quickly. Finally, this is also supported by the fact that the ETAS operating system ERCOS^{EK} has a precompiled library. As a result, hardly any effort is required when making changes to the operating system configuration except for the link process.

Modeling the Control Process

Specifying complex control algorithms without modeling the related process is no longer conceivable today. ASCET offers a special advantage: The control process and the process model can be developed in the same system. This does away with extra time consumed by conversions or simulator couplings. In small projects, the process and the control stem from a single source, i.e. they are developed by a single engineer.

ASCET also has the openness of integrating other systems. For example, ETAS has created a link to Matlab which permits the design engineer to simulate Simulink models together with ASCET specifications.

All in all, this technology takes a giant step forward in the direction of efficiency and effectiveness. This applies above all when the laboratory car is available in the form of models and hardware. It can save a lot of time. Safety-

critical analyses can be speeded up from the desk at low cost and without danger. The driver need only step into the car for the trials. ETAS offers this technology now for many systems.

4.3.2 Implementation and Code Generation

Automatic code generation for series control units is the key to an efficient development methodology. ASCET has set standards here. It is not only the special challenge of fixed point arithmetic. Operating system interfaces must also be configured for floating point processors. Memory management must be optimized and hardware encapsulation must be integrated. Furthermore, a distinction must be made in floating point systems between precise and double precise data. This is at least important in the simulation.

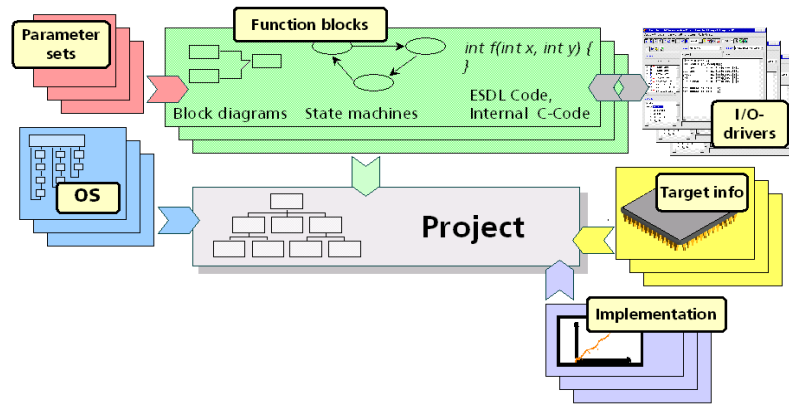


Fig. 4-20 Principle of automatic code generation in ASCET: the inputs

In ASCET, the implementation consists of the data types, value ranges and memory storage information. Conversion equations form the link between these implementation levels and the physical data description. In addition to memory storage information, there is a definition whether functions are calculated in line or whether a utility should be used for analyzing characteristics.

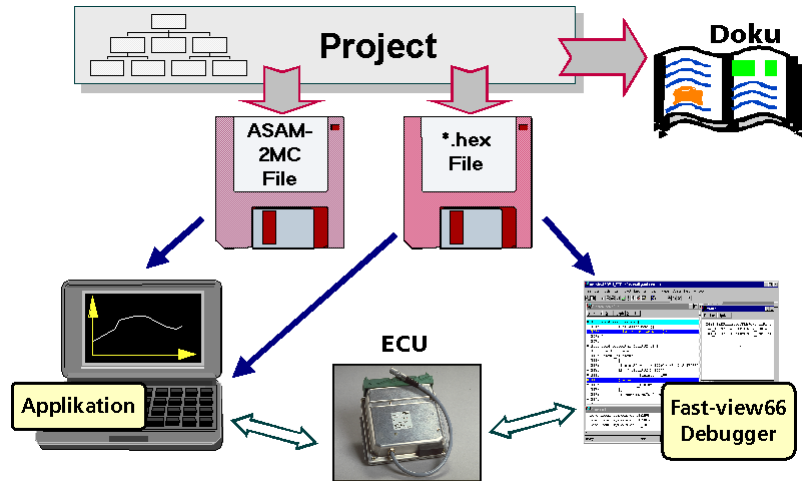


Fig. 4-21 Principle of automatic code generation in ASCET: the outputs

Automatic code generation is not a one-way street. In parallel to the generator C code, an ASAM-MCD-2MC description is required. It provides the necessary address information for application and measuring systems. This is obtained by reading back and interpreting the MAP file after generating the program. ASCET-SE contains this function. You can even start from an ASAM-MCD-2MC file, read it into ASCET and base a first data model in ASCET from it (re-engineering an existing program).

Algorithms

The automatic conversion of block diagrams, text specifications and state machines in target-specific C code runs over a common intermediate layer that uses implementation information to optimize the mapping of algorithms on the target system. In addition to mastering complex logic, the arithmetic poses a special challenge here. The same goes for integer code generation. Even the simple assignment

$$a = b$$

of two variables is not a trivial operation for code generation if the implementations are different. Let a and b be implemented by the following equations as unsigned 8 bit variables (range from 0 to 255):

$$a = 2 * a_impl, b = 3 * b_impl$$

This is followed by a simple substitution:

$$a_impl = 3 * b_impl / 2$$

Care must be taken here with the series of operations in order to consider the requirement for maximum precision. If you first perform the division, the various conversion equations would be ineffective due to the integer computation and the results would be about 50% incorrect.

$$a_impl = (3 / 2) * b_impl = b_impl$$

The question of overflow must be taken into account. This means that if you first multiply by 3, there is an overflow as soon as b_impl becomes greater than $255 / 3 = 85$. Similarly, you must always be careful of underflows and rounding errors. If you first divide by 2, this is equivalent to a right shift operation, i.e. the last bit is dropped. No distinction can then be made whether b_impl has the value 1 or 0. In both cases, the result for a_impl and thus also for a is the value 0.

In fact, the assignment $a = b$ only makes sense if the physical ranges are identical (here max. 0 to 510). b_impl can therefore assume the maximum value $510 / 3 = 170$. An overflow can occur here and must be avoided at all costs. You may then think of making a case distinction in the code generation, i.e. first multiply for values from b_impl to 170 and first divide for values from b_impl greater than 170. But this leads to a requirement for more code. So here, you must accept a negligible error in precision of max. 1.5. within the entire value range.

It is clear that the situation itself can become more difficult with regular arithmetic operations with few operands, not to mention complex links and expressions. The automatic code generation feature in ASCET relieves the user from this type of problem.

Memory Handling

The control unit architecture defines memory areas which serve a variety of purposes. Some areas are reserved for the program, others store applicable data. A physical distinction must be made between ROM, RAM and Flash. In many control units, special areas are reserved for storing bits. In many cases, values can only be stored to even addresses. Some memory areas are also reserved as inputs or outputs to and from the processor periphery. ASCET considers this information continuously. The control unit architecture is available to the user to describe the implementation and it can be adapted to the physical conditions in every case.

All standard data (e.g. measurement and calibration variables) automatically receive a default memory area assigned. The automatic code generation function transfers these implementations to pragma statements in C code. Here again, the user is relieved of annoying, error-prone management work.

Operating System Configuration

Configuring the operating system with ASCET is also very simple for the user. The user is guided by a graphical user interface to specify the necessary information in the ASCET project. C code is also generated automatically.

The main task of automatic code generation is to optimize communication between various processes by means of messages. If interruptions occur, all messages must be saved in order to continue working with consistent data after the interruption. This requires an enormous effort which is frequently superfluous. In fact, only data which can be inconsistent during an interruption need be saved. This provides a high degree of potential optimization that is exploited in the automatic code generation function.

As a result, the user saves a lot of work since the complex interruption options are analyzed automatically and are considered in the code generation function. It excludes errors caused by forgetting a backup operation. The greatest relief from the burden of complex management operations come during the development phase when the structure of processes and tasks can change. ASCET offers both convenience and reliability here.

Platform Dependence and Project-specific Adaptation

The high performance of ASCET and automatic code generation are not only reflected in successful product developments which our customers have already carried out, but also in the capability of mastering future requirements effortlessly. The key to this is the open, adaptable interface of this technology. The expanded applicability has been proven by many instances, e.g. integrating non-proprietary operating systems, successful transfer of project-specific requirements (e.g. in the form of naming rules or adaptation to customer-specific development processes).

Dividing the development into physical modeling and the specification of the implementation forms the basis for extensive adaptation possibilities. The platform-specific characteristics can be encapsulated at a central point (of the implementation) and can therefore be exchanged in a single step. To this is added the wide configuration options of the automatic code generation function. Building on a central intermediate layer in ASCET, code production rules are used in **ASCET-SE** and they are adaptable to specific requirements. This gives ASCET unprecedented access to the automatic code generation function, if this is necessary or useful during development activities.

4.3.3 Prototyping with ASCET

ASCET produces the first results very quickly and they can be transferred just as quickly into products. The factors governing the success of the prototyping feature is the experimental environment and the supported hardware. This section describes this in more detail.

Experimental Environment for Extreme Requirements

The ASCET experimental environment lets the user carry out various analyses on the development objects without changing them. It is therefore a universal working environment in which to define and generate stimuli quickly and simply, record and analyze data, and change variables. The simulation can be started or stopped at any point. Settings obtained, optimized datasets and window configurations can be saved separately for re-use as a so-called experiment and therefore re-used in the long term. Several experiments can be managed for each model for a variety of purposes. There are several high-performance graphical and text displays and editors for measuring and adjusting and they can be combined flexibly. The ASCET experimental environment offers this excellent functionality both offline or online in the real-time experiment.

After the first analysis of the control algorithms as a physical experiment, ASCET also supports the analysis of the quantization and implementations in the same working environment. In this way, the development can be refined step-by-step through to the actual product. Special support is also given by providing special monitor variables in the experimental environment.

During the first analysis of new modules or classes, the experimental environment permits the step-by-step inclusion of single processes and methods in the simulation. This dynamic operating system configuration directly in the experimental environment permits a flexible, fast operating method and results very quickly in realistic results.

The viewing and adjusting elements used in the ASCET experimental environment are identical to those used in our INCA-PC application and measuring system. It ensures the continuity of the development environment at the operating system level from the concept through to the product.

Simulation Systems: Hardware and Software

Prototyping mainly lives from the hardware supported. The main part is the execution platform for the simulation. ETAS offers a scalable concept here. A high-performance computer node (PowerPC) performs the basic analysis of new control algorithms. This platform has proved itself and offers a variety of expansion options.

The operating system on the simulation platform is ERCOS^{EK} as in the series control units. Prototyping is therefore highly realistic and all the necessary analyses can be performed in real-time.



Fig. 4-22 ES1000 Universal VME bus System

Many cards can be included in the simulation as plug-in cards in our ES1000 VME bus system. The system therefore supports the main industry standard, The possibilities cover the use of existing commercial cards for including external sensors and actuators through to the use of special project-specific signal conditioning cards that are made to measure.

Integrating Existing Sensors and Actuators in a Closed Control Loop

ETAS offers hardware and software solutions for all customary interfaces to sensors and actuators. We have VME bus cards for

- AD/DA conversion
- PWM signal exchange
- CAN interface
- LIN interface
- Digital I/O
- FPGA

In addition we have special solutions for temperature measurement and recording emission data. The solutions can be easily integrated in this architecture.

The PC interface is implemented by ethernet. It can therefore work with mobile laptops. Older hardware system using a parallel (printer) port are still supported, as well. An efficient protocol exchanges data between the experi-

mental hardware and experimental environment online in ASCET. This does not disturb the execution of the closed local loop at hardware level. Working with ASCET ranges from the concept to the realistic prototypes.

4.3.4 Bypass

We provide two physical interfaces for the use of our bypass technology: Memory emulation via ETK and the CAN interface. Other hardware interfaces can be customized on request. All bypass systems are based on an executable control unit for which a function can be changed or developed. In order to apply the bypass, there must be a modification of this function. The modification contains the function scheduling. In other words, the time scale in which the function is processed must be known in advance. Depending on the situation, the necessary safety precautions must be taken in case communication errors occur. For example, if a timeout occurs in the CA communication, an emergency program can be started or default data must be anticipated.

Bypass technology is mainly intended as a support for development joint ventures where certain functions cannot be published for reason of know-how protection. Since only one interface is freed up, design engineers can save other more complex measures. What is more, a major part of the total system is always stable and partners can fully concentrate on developing an innovative function. The result is a cost-effective solution with the greatest utility.

ETK Interface— Memory Emulation

The emulator test probe (German acronym: ETK) is a unique interface to the control unit developed by ETAS. It is based on the full or partial emulation of the control unit memory in the form of a DPRAM. The control unit has direct access to it from one side and from the PC side, it can be operated transparently and read out. This requires a change to the control unit since the memory emulation can only be implemented over very short distances between the ETK and the processor. Today ETKs are already very small and can be designed in hybrid technology.

For our ES1000 VME system, we offer plug-in cards for connection to the ETK. Since ETKs are offered or modified for many different processors, this is a universal solution for the application.

CAN Interface

In cases where an ETK is not used for various reasons, the solution is a CAN interface. This requires no change to the control unit hardware and is a possible solution in almost all cases. However, this solution does not attain the performance data of the ETK. In addition, a sufficient number of free messages

must be available. Given these constraints, the CAN bypass in conjunction with ASCET is still the first choice in many cases since it is particularly cost-effective, not to speak of its characteristics already described.

4.3.5 Reuse and Open Interfaces

The topic of reuse has been discussed for many years with keywords such as modularization, object orientation and interface compatibility. In ASCET, reuse and open interfaces have become practical reality. This is the result of the careful analysis of development steps and development results in real projects and the direct transfer of these observations into useful characteristics in our tools.

This is how we have put together a library of excellent blocks for specifying control algorithms. These blocks have proved themselves in practice and represent the standard for our competitors.

Reusable experiments are linked to this library. They permit a reproducible response for old and new development tasks. Our interfaces to tools for application, trials and testing can include all the relevant work steps in the development of the overall system. Not only do we offer interfaces to other ETAS tools but also interface standards such as ASAP and MSR which are universally accessible.

Reuse through Database Support

The ASCET database system is the basis for our reuse concept. It was implemented in the ASCET kernel from the start and thus forms the backbone of the tool. The benefits for the user are obvious. All data is saved reliably in a separate work environment and is protected from unauthorized or unintended manipulation or even destruction. In the end this data represents valuable work results which require special protection. To manage this data, the ASCET database provides all the necessary mechanisms for efficient day-to-day work both for single users as well as for teamwork.

Program and Database Management Using Configuration Management Tools

The use of purchased configuration management tools has frequently been preferred in order to manage variants of development streams or to permit large teams to work together on complex solutions. These tools also manage other company data. In fact, tool selection is specified by the customer. To be able to interface any configuration management tool, ASCET supplies the necessary interfaces. In this solution, ASCET operates as a client to the configuration management tool. However, if ASCET is used as a server, ASCET's complete database functionality can be harnessed for a detailed modification

to the management mechanisms of the configuration management tool. All that has to be done is to modify the interfaces in JAVA. ASCET can then be used to implement innovative and complex control tasks.

If you intend to use a configuration management tool, please contact ETAS for a solution adjusted to your particular requirements.

4.4 ASCET Software Structure

The ASCET software family is divided in four products, which support the different phases in the users' working process.

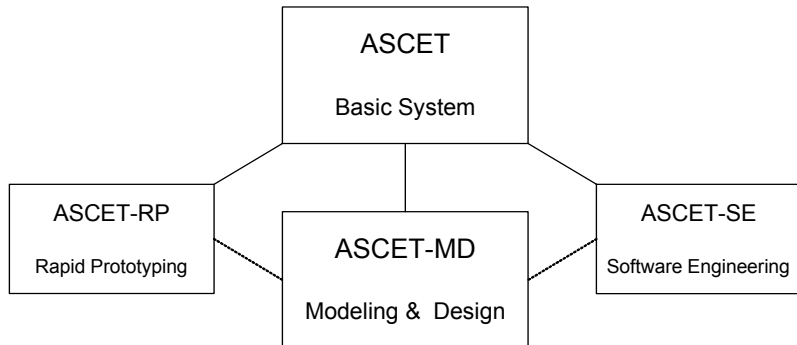


Fig. 4-23 The modular structure of ASCET

The following sections briefly describe the functional scope of the individual products.

ASCET Basic System

The ASCET base system is the foundation for the other products, which cannot be installed without it.

ASCET Modeling & Developing

ASCET-MD allows the specification of models as block diagrams, in ESDL, or in the C programming language. As in previous ASCET-SD versions, models can be specified and managed, as well as simulated in offline experiments.

The same version of ASCET-MD can be installed only once on a given PC.

ASCET-RP

ASCET-RP offers the full functionality required for rapid prototyping. It is described in a separate manual.

With ASCET-RP, you can view components; modeling or changing models or model elements, however, is possible only in connection with ASCET-MD.

The same version of ASCET-RP can be installed only once on a given PC. It is possible to install ASCET-RP, ASCET-MD, and ASCET-SE for one or multiple micro controllers simultaneously.

ASCET-SE

ASCET-SE offers the full functionality required for the generation of ECU code. It is described in a separate manual.

With ASCET-SE, you can view components; modeling or changing models or model elements, however, is possible only in connection with ASCET-MD.

ASCET-SE is available in ports for several micro controller targets. Unlike the other products of the ASCET family, ASCET-SE can be installed simultaneously for different targets on the same PC. It is possible to install ASCET-SE for one or multiple micro controllers, ASCET-MD, and ASCET-RP simultaneously, as well.

5 **General Operation of ASCET**

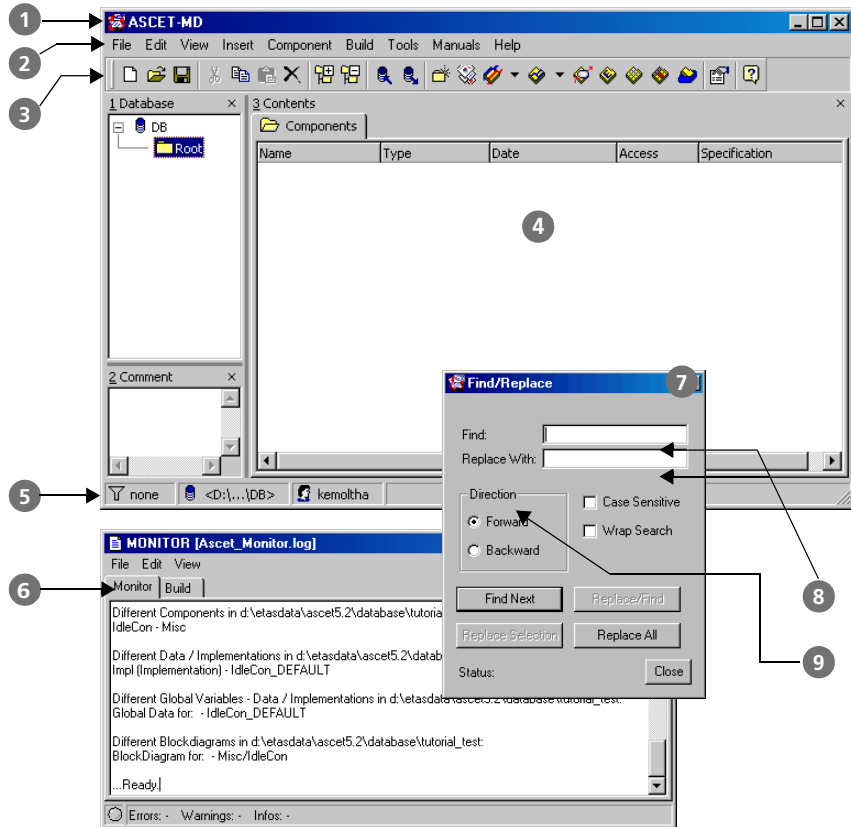
This section provides information on the window and menu structures, control options using mouse and keyboard, and help features.

We encourage you to read this chapter since some of the control options are described only here. Although all techniques explained are Windows standards, they might be unknown to a less experienced Windows user. They are therefore described here as a central summary.

Simple operation using the keyboard has been emphasized during the development process of ASCET. For special features and deviations from Windows conventions regarding keyboard operation, please refer to "Keyboard Control" on page 219.

5.1 Window Structure

The ASCET window elements



- Title bar (1)
- Menu bar (2)
- Button bar (3)
- Window area (4)
- Bottom bar (5)
- Tab (6)
- Dialog box (7)
- Fields (8)

- Field caption (9)

5.2 Button Bars

The most common commands are also available as buttons. This way, a command can simply be executed with a click on the mouse.

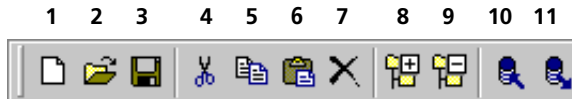
note

All commands which can be executed using the individual buttons are also provided in the corresponding menus.

All buttons located on the button bar are mouse-sensitive. If you place the cursor on a button and hold it for one second, a text box is displayed right next to the button selected which displays the button function.



5.2.1 Buttons in the Component Manager



1. New (creates a new database)
2. Open
3. Save
4. Cut
5. Copy
6. Paste
7. Delete
8. Expand all
9. Collapse all
10. Import
11. Export



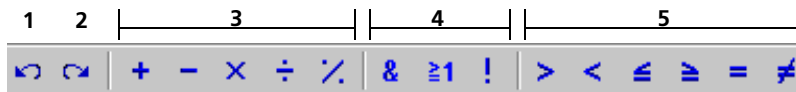
- 12. Insert Folder
- 13. Insert Project
- 14. Insert Module - <Type>
- 15. Insert Class - <Type>
The arrows (14a and 15a) can be used to select the object type.
- 16. Insert State Machine
- 17. Insert Enumeration
- 18. Insert Boolean Table
- 19. Insert Conditional Table
- 20. Insert Container
- 21. Options
- 22. ? (opens the "AboutASCET" window with information on the installed products of the ASCET product family)



23a

- 23. Search - <criteria> (searches the database for a search string, using a defineable search criterion)
The arrow 23a can be used to select the search criterion.
- 24. Input field for the search string

5.2.2 Button Bars in the Block Diagram Editor



- 1. Undo
- 2. Redo
- 3. arithmetic operators (Addition, Subtraction, Multiplication, Division, Modulo)

4. logical operators (And, Or, Not)
5. comparison operators (Greater, Less, Less or Equal, Greater or Equal, Equal, Not Equal)

6 7 8 9 10 11 12 13 14 15 16 17



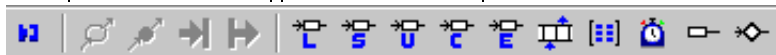
6. Abs (returns the absolute value of the input)
7. Max (returns the largest input)
8. Min (returns the smallest input)
9. Between (checks whether the input lies between the limiting values)
10. Negation (reverses the input sign)
11. MUX
12. Case
13. If-Then
14. If-Then-Else
15. While
16. Switch
17. Break (specifies immediate exit from a process/method)

18 19 20



18. combo box to select the number of operator inputs
19. combo box to select a view
20. combo box for the zoom factor

21 22 23 24 25 26 27 28 29



21. Connect
22. elements in state machines (State, Junction, Input, Output), only available when you edit a state machine
23. variables (Logic, Signed Discrete, Unsigned Discrete, Continuous)
24. Enumeration

- 25. Array
- 26. Matrix
- 27. dT system parameter

note

The name dT is reserved for the system parameter. You cannot create any other element with the name dT . Since upper and lower case letters are not distinguished, the names DT , $d\tau$, and $D\tau$ are reserved, too.

- 28. Continuous Parameter
- 29. Implementation Cast



- 30. characteristic lines and maps (Distribution, One D Table Parameter, Two D Table Parameter)
- 31. combo box to select the type of the characteristic line/map
- 32. Resource
- 33. messages (Receive, Send Receive, Send; only available for modules)
- 34. literals (String, True, False, 0.0, 1.0)
- 35. Self (reference to the current object itself)



- 36. Hierarchy
- 37. Comment
- 38. Generate Code
- 39. Open Experiment for selected Experiment Target

Unnumbered button bar elements are always disabled in the block diagram editor.

5.2.3 Button Bars in the C Code and ESDL Editor



The button bar elements (1) to (9) are available both in the C code editor and in the ESDL editor. Button (7), however, is deactivated in the C code editor. They correspond to the button bar elements (23) to (31) in the block diagram editor (cf. page 85).



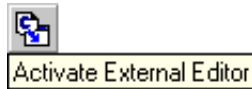
Buttons (10), (11), (13) and (15) are available in both editors, too. They correspond to the buttons (32), (33), (38) and (39) in the block diagram editor (cf. page 86).

12. External Source Editor (C code editor only; opens the editor for external sources)

14. Compile Generated Code (activated in the C code editor only)

Unnumbered button bar elements are always disabled in the C code and ESDL editor.

16



16. Activate External Editor (in the bottom bar of the respective editor; activates the possibility to edit the component code in any text editor outside ASCET).

5.2.4 Button Bars in the CT Block Editors

CT blocks can be specified in C code, ESDL or as block diagrams. The following buttons are available in all three editors:



1. Input

2. Output
3. Constant
4. combo box to select the parameter type
5. One D Table Parameter (characteristic line)
6. Two D Table Parameter (characteristic map)



The button bar elements (7) and (9) correspond to the elements (38) and (39) in the block diagram editor (cf. page 86).

8. Compile Generated Code (activated in the C code editor for CT blocks only)

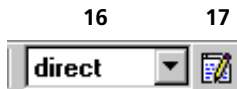
Unnumbered button bar elements are always disabled in the CT block editors.

The following buttons are only available in the ESDL or C code editor for CT blocks:



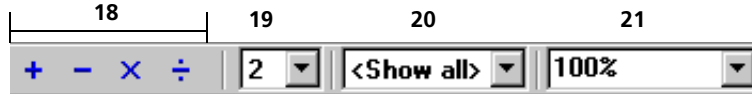
10. Discrete State
11. Continuous State
12. Steplocal (variable)
13. Parameter (type is selected in combo box (4))
14. Dependent Parameter (type is selected in combo box (4))
15. Activate External Editor (in the bottom bar of the editor; activates the possibility to edit code in any external text editor)

The following button bar elements are only available in the C code editor for CT blocks:



16. combo box to select direct or non-direct block behavior
17. External Source Editor (opens the editor for external sources)

The following buttons are only available in the block diagram editor for CT blocks:

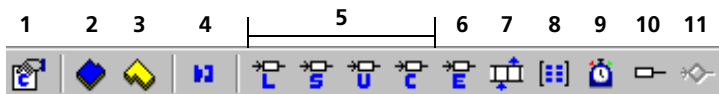


- 18. arithmetic operators (Addition, Subtraction, Multiplication, Division)
- 19. combo box to select the number of operator inputs
- 20. combo box to select a view
- 21. combo box for the zoom factor



- 22. Connect
- 23. Global Parameter (type is selected in combo box (4))
- 24. Hierarchy
- 25. Comment

5.2.5 Button Bar Elements in the Project Editor



- 1. Specify Code Generation Settings (opens the "Settings" dialog with the code generation settings)
- 2. Edit Data (opens the data editor for the project)
- 3. Edit Implementation (opens the implementation editor for the project)

The button bar elements (4) to (11) correspond to the button bar elements (21) and (23) to (29) in the block diagram editor (cf. page 85).



The button bar elements (12), (13), (15) and (16) correspond to the button bar elements (30), (31), (36) and (37) in the block diagram editor (cf. page 86).

- 14. Send Receive Message

Unnumbered buttons are always disabled in the project editor.



Button (17) corresponds to button (38) in the block diagram editor (cf. page 86).

18. Compile Generated Code

19. Build Executable Code

20. Rebuild Executable Code

21. Transfer Project to selected Experiment Target (transfers the project to the experiment selected in combo box (25))

The button is only enabled when you have selected the target `ES1130` or `ES1135` from the target options and the entry `INCA` or `INTECRIO` from combo box (25).

22. Open Experiment for selected Experiment Target (generates code and opens the experiment selected in combo box (25))

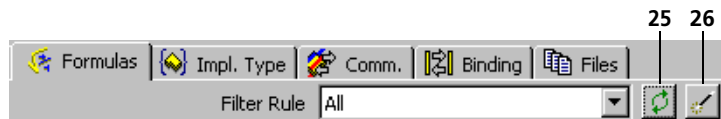
The button is *disabled* when you have selected the entry `INTECRIO` or `INCA` from combo box (25).

23. Reconnect to Experiment of selected Experiment Target (restores the connection to the experiment running on the target selected in (25))

The button is *disabled* when you have selected the target `ES1130` or `ES1135` from the target options or the entry `INCA` from combo box (25).

24. Experiment Target

The available entries depend on the target you chose in the target options and on the other programs (e.g., `INTECRIO` or `INCA` installed on your PC.

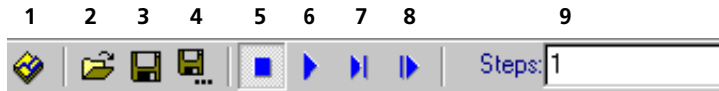


25. Refresh for the "Formulas" tab

26. Add missing Formulas

Finds undefined formulas in the active implementation, and adds a linear formula behaving like the identity for each detected formula name.

5.2.6 Button Bar Elements in the Offline Experiment



1. Exit to Component (closes the experiment and opens the component editor; the appearance of the button depends on the component used in the experiment)
2. Load Environment (loads an experiment environment, i.e. predefined measure and calibration windows with assigned variables)
3. Save Environment
4. Save Environment As
5. Stop Offline Experiment
6. Start Offline Experiment
7. Pause Offline Experiment (pauses the experiment; continue with buttons (6) or (8))
8. Step Offline Experiment
9. input field for step size



10. Open CT Solver
This button is *only* available when you are experimenting with a CT block or a hybrid project.
11. Open Event Generator
This button is *not* available when you are experimenting with a CT block or a hybrid project.
12. Open Event Tracer (opens the "Event Tracer" window for data tracing)
13. Open Data Generator
14. Open Data Logger
15. Update Dependent Parameters
16. Expand / Collapse Window (shows/hides the component display)
17. Always on top (keeps the "Physical Experiment" window always in the foreground of the monitor)

18. Navigate down to child component

19. Navigate up to parent component

5.3 Operation Using The Keyboard

Simple operation using the keyboard has been emphasized during the development process of ASCET. Individual keys are preferred over the function keys <F1> to <F12>, which in turn are preferred over keyboard shortcuts using <CTRL> and <ALT>. You can display a complete overview of the keyboard commands currently used at any time by pressing <CTRL> + <F1>.

5.3.1 General Keyboard Control

This table lists the most important keys and keyboard shortcuts used to control the New Experimentation Environment of ASCET. For a complete list of all keyboard commands, please refer to chapter "Keyboard Control" on page 219.

Key	Function
<F2>	change to Edit mode (e.g., for table entries)
<SHIFT> + <F10>	open context menu for selected item (right mouse button)
<ALT>	activate main menu
<ALT> + <F4>	close active window; in the Component Manager: exit ASCET
<ALT> + <F6>	switch between open ASCET windows
<ALT> + <SPACE>	open system menu of application window
<ALT> + <TAB>	switch between open applications
<CTRL> + <F1>	show hotkey assignment
<CTRL> + <A>	select all items (e.g., in a list)
<CTRL> + <C>	copies data to the clipboard
<CTRL> + <V>	inserts data from the clipboard
<CTRL> + <X>	cuts data to the clipboard
<CTRL> + <Y>	redo last action (only in editors)
<CTRL> + <Z>	undo last action (only in editors)
<DOWN ARROW> (↓), <UP ARROW> (↑), <LEFT ARROW> (←), <RIGHT ARROW> (→)	move cursor to table item or list item with arrow keys, <RIGHT ARROW> also opens folder, <LEFT ARROW> also closes folder,
<ENTER>	confirm input and quit input mode; expand or collapse branches

Key	Function
	delete selected item
<ESC>	cancel input, discard changes
<SPACE>	select table or list item or deselect active selection
<TAB>	move focus to next item (option) in a window (<SHIFT> + <TAB>: opposite direction)

5.3.2 Keyboard Control According to the Windows Conventions

The WINDOWS® conventions apply to the general operation of ASCET, such as navigating through menus or activating a specific window.

Pressing the underlined letter in a menu while holding down the <ALT> key activates the corresponding command. You can activate a subordinate menu command by pressing the underlined letter together with the <SHIFT> key.

For example, to open the **File** menu in Component Manager with a keyboard command, press the <ALT> + <F> key combination.

To switch to the next window or list box within the working windows, press the <TAB> key (in the order from top left to bottom right). As an alternative, you can use <ALT> key and the underlined character (number or letter) of the field or list label to switch to the corresponding field or list box.

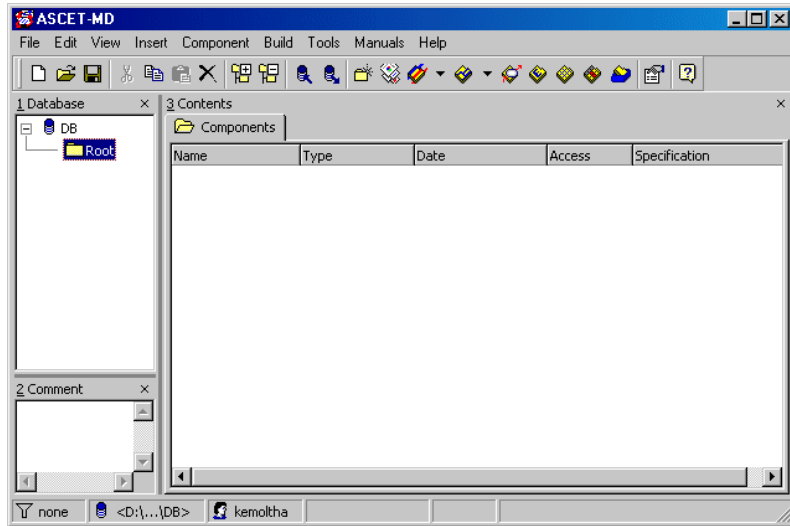
The arrow keys allow you to skip to the next item in list boxes. You can select multiple items by making your selection while pressing the <SHIFT> key.

According to the WINDOWS® conventions, you can switch between the working windows using the keyboard shortcut <ALT> + <TAB>. In this sense, all subsystems of ASCET, such as the component editors, the experiment environment, or implementation or data editor, are treated as individual applications.

If you need <TAB> within the window for other functions, e.g. when editing text, you can switch to the next window element using the keyboard shortcut <CTRL> + <TAB>.

Switching between several tabs in a window or field (e.g., in the Component Manager, "3 Contents" field) is done—pursuant to the MS-WINDOWS® convention—by pressing the <CTRL> + <TAB> key combination.

Within a window, you can use the underlined letter of the box or list title, while holding down the <ALT> key, to switch to the corresponding list box. For example, <ALT> + <2> would activate the "2 Comment" text field.



5.4 Operation Using The Mouse

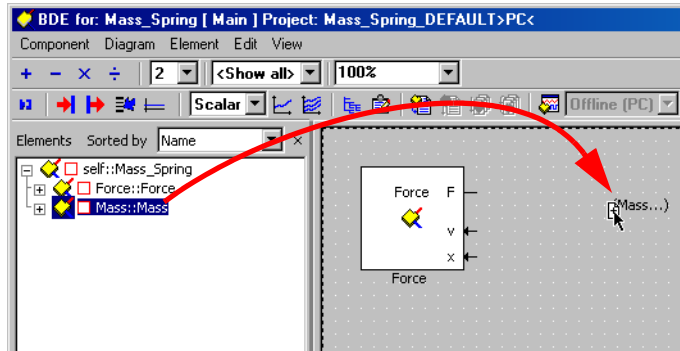
In the office or lab, you can use the mouse to conveniently control ASCET. The use of the mouse corresponds to WINDOWS® conventions.

You can select multiple items by making your selection while pressing the <SHIFT> or <CTRL> key.

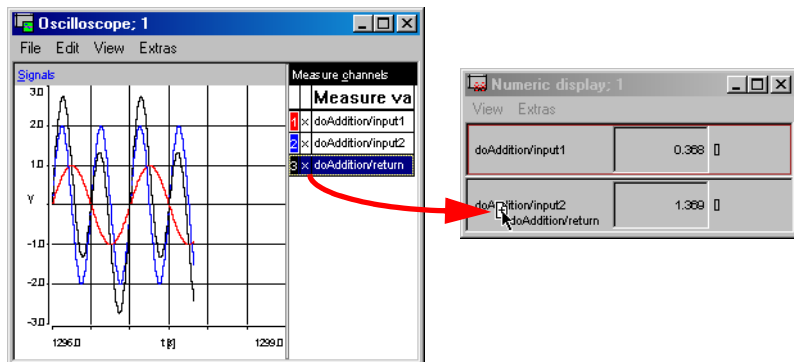
Right-clicking on the window elements opens respective context menus.

5.4.1 Drag & Drop

When creating block diagrams, or setting up measure or calibration windows, you can use the drag and drop technique (left-clicking on the variable, keeping the mouse button pressed and moving the variable to the target window/field using the mouse).



It is as easy to copy a value from one window to another. Commands which cannot be performed, such as copying a parameter to a measuring window, will be ignored.



5.5 Hierarchy Trees

ASCET often displays information such as the contents of a database in a hierarchical tree structure. In order to see all branches and the entire contents of such a tree structure, you have to expand or collapse the branches. ASCET allows you to expand several branches automatically or to expand specific partial trees.

To expand several branches automatically:

- To expand or collapse all branches at once, you can use the **View → Expand All** or **View → Collapse All** menu option or the buttons in the button bar (see „Buttons in the Component Manager“ on page 83).

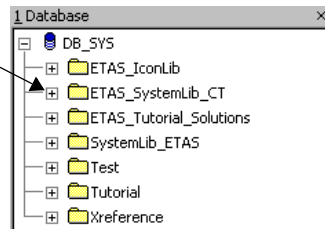
To expand individual partial trees:

- To open the desired branch, click with the mouse on the little "+" box next to this item or press the <+> key. Clicking a second time on the same little box or pressing the <-> key collapses the branch again.

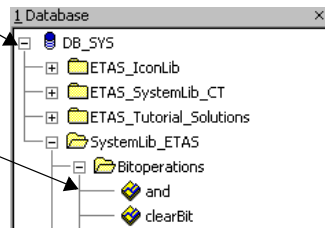
Or

- Move the focus to the item with the <UP/DOWN ARROW> key and then press the <RIGHT ARROW> key to expand the branch. Collapse the branch by pressing the <LEFT ARROW> key.

A "+" box indicates a branch that can be expanded.



A "-" box indicates an expanded branch.

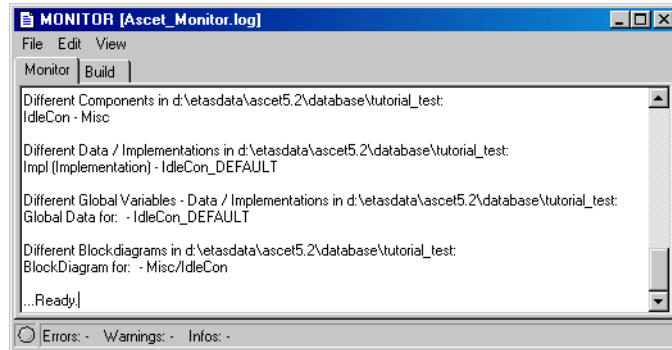


If there is no box, this is the end of the structure. This branch cannot be further expanded.

5.6 Supporting Functions

5.6.1 Monitor Window

The monitor window (see chapter 2.4 in the ASCET user's guide) is used to log the working steps performed by ASCET. All actions, including errors and notifications, are logged. As soon as an event is logged, the monitor window is displayed in the foreground.



In addition to displaying information, the monitor window also provides the functionality of an editor.

- The display field in the "Monitor" tab of the monitor window can be freely edited. This way, your own notes and comments can be added to the ASCET messages.
- The ASCET messages can be saved as text files along with your comments.
- Other ASCET text files already stored can be loaded so that you can compare specific working steps.

5.6.2 Keyboard Assignment

You can display an overview of the keyboard commands currently used at any time by pressing <CTRL> + <F1>.

5.6.3 Manual and Online Help

If not specified otherwise during installation, the entire ASCET manual is available electronically and can be displayed on the screen at any time, e.g., via the menu options in the **Manuals** menu. The volumes, named **ASCET V5.2**

Quickstart.pdf, **ASCET V5.2 Manual.pdf**, **ASCET V5.2 Reference.pdf**, are stored in the ETAS\ETASManuals folder. Printed manuals can be ordered here:

http://www.etasgroup.com/order_manual/ascet

Using the index, full text search, and hypertext links, you can find references fast and conveniently.

The online help can be accessed via the <F1> key. It is stored in the ETAS\ASCET5.2\Help folder.

6 **Tutorial**

The tutorial mainly addresses users who are new to ASCET. It describes the use of ASCET using practice-oriented examples. The entire tutorial contents are subdivided into short individual components based on each other. Before you start working on the tutorial, you should have read Chapter "Understanding ASCET" on page 47.

6.1 A Simple Block Diagram

In ASCET you use components, such as classes and modules, as the main building blocks of your applications. You can either use predefined components, which come with ASCET or have been developed earlier, or create your own, which is what you will be doing in this tutorial.

In ASCET components are usually specified graphically. Once all the components have been specified, they are assembled into a project, which forms the basis of an ASCET software system. A software system consists of C code that has been generated from the graphical model description, and which can be run on a microcontroller or experimental target computer.

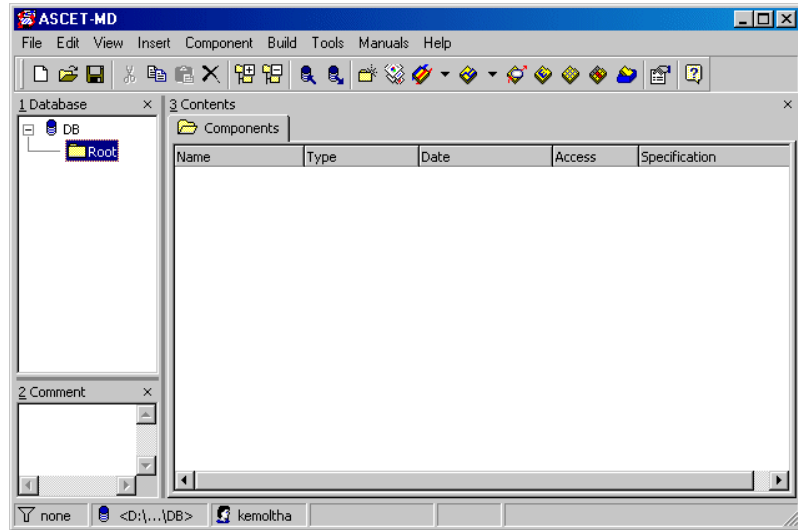
6.1.1 Preparatory steps

Before you can start, you have to open a database to work in. All the components of this tutorial will be stored in this database, so you will only have to do this once.

All components and projects for this tutorial can be found in the folder called `ETAS_Tutorial_Solutions` in the database `tutorial`. It is therefore not necessary to specify all the components described here yourself .

It is, however, advisable to specify at least the components of lessons one, three and four, to get some practice using ASCET.

At the start of ASCET, the Component Manager opens, loading the database that was last opened.



It is recommended that you create a new database for the tutorial to keep the data transparent.

To create a new database:

- In the Component Manager, select **File** → **New Database**

or



- click on the **New** button

or

- press <CTRL> + <N>.

The "New database" window opens.



- Enter the name `Root`.
- Click on **OK**.

The new database, containing only the database name and the `Default` folder, opens.

To open a database:

When the `tutorial` database already exists, proceed as follows:

- In the Component Manager, select **File** → **Open Database**

or

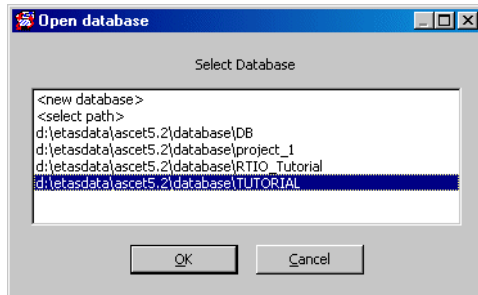


- click on the **Open** button

or

- press `<CTRL> + <O>`.

The "Open Database" dialog box is displayed. It contains a list of the databases in the current database path.



- If the `tutorial` database is on the list, select it and click on **OK**.

The Component Manager displays the contents of the `tutorial` database.

- If the `tutorial` database is stored somewhere else, use the `<select path>` option to specify the database and click on **OK**.
- In the "Select database path" window, select the database and click on **OK**.

The first step in creating your own components is to create a new top level folder named `Tutorial` and a subfolder named `LessonN` for each lesson.

To create a new folder:

- In the "1 Database" field, select the database name.
- Select the menu item **Insert** → **Folder**

or



- click on the **Insert Folder** button

or

- press <INSERT>.

A new top-level folder named `Root` appears in the "1 Database" pane.

- Change the name of the top-level folder to `Tutorial`. You can type over the highlighted name and then press <ENTER>.
- Select the folder `Tutorial`.
- Select **Insert** → **Folder** once again.

A new folder named `Folder` is created in the "1 Database" pane.

- Change the name of the new folder to `Lesson1`.

All the components you create in this tutorial will be stored in the `LessonN` folder. You should create a new folder for every lesson. Every database has at least one top-level folder which can have any number of subfolders.

Note

In ASCET 5.0, all folder and item names and the names of variables and methods they contain must comply with the ANSI C standard.

You can proceed by creating your first component in the `Lesson1` folder.

To create a component:

- In the "1 Database" pane, click on the folder `Lesson1`.
- Select **Insert** → **Class** → **Block Diagram**.
A new component named `Class_Block-diagram` appears in the "1 Database" pane under the `Lesson1` folder. This component is of type *class*, which is frequently used in ASCET.
- Change the component name to `Addition`.

6.1.2 Specifying a Class

After you have created a new component in the `Tutorial/Lesson1` folder you can specify its functionality. First define the interface for the component, i.e. its methods, arguments and return values. Then draw a block diagram that specifies what the component does.

To specify the functionality of a component:

- In the "1 Database" pane, select the component `Addition`.
- To open the component, select **Component** → **Edit**

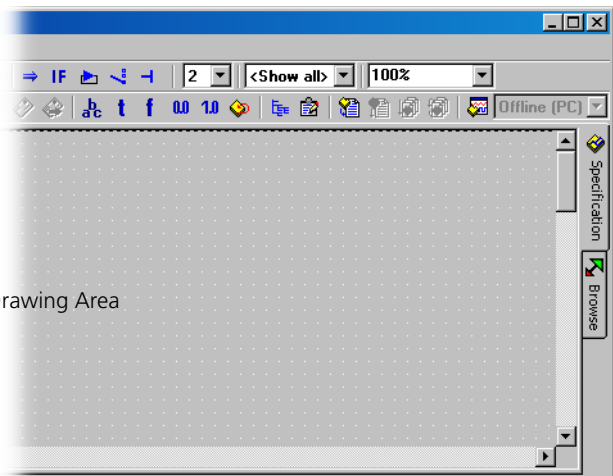
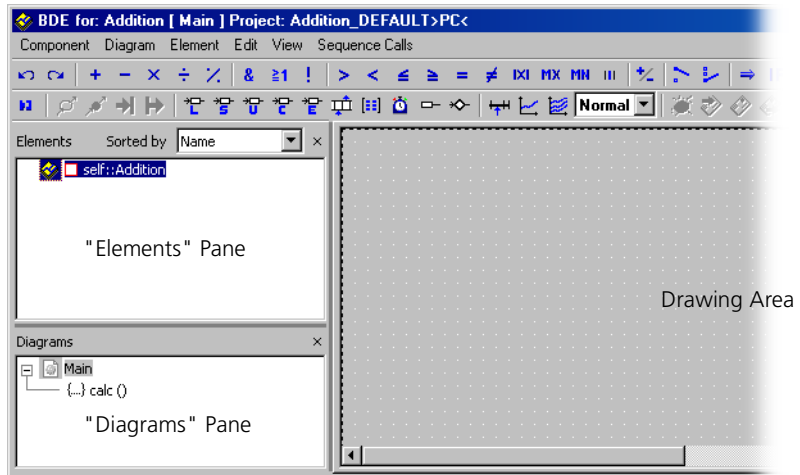
or

- double-click on the component

or

- press <RETURN>.

The block diagram editor opens. This is the main window for specifying component functionality.

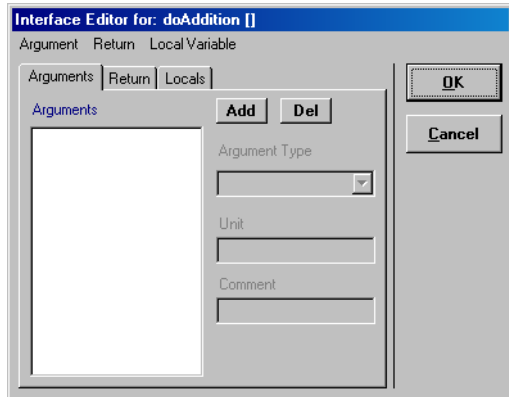


- In the "Diagrams" pane, select the method `ca1c`. This method is created by default.
 - Select **Diagram** → **Rename**
- or
- press <F2>.
- The name of the method `ca1c` is highlighted.

- Change the name of the method to `doAddition`.
- Select **Diagram** → **Edit**

or

- double-click on the method name.
The interface editor for the method opens.



Every class needs at least one method. Methods in ASCET are similar to methods in object-oriented programming, or functions in procedural programming languages. A method can have several arguments and one return value (these are all optional). Arguments are used to transmit data to a component. Return values are used to return results of calculations within the component to the "outside".

To specify the method interface, you can add two arguments of type `continuous` and a return value using the interface editor.

To specify the method interface:

- In the interface editor, select **Argument** → **Add**.
A new argument called `arg` appears in the "Arguments" pane.
- Change the name of the argument to `input1`.

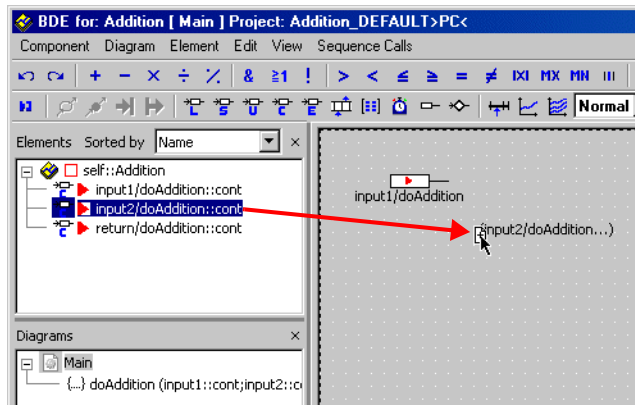
- Add another argument called `input2`.
By default the data type of the arguments is set to *continuous* (or *cont* for short), which is what you need in the example.
- Activate the "Return" tab of the interface editor.
- Check the **Return Value** tick box.
The type of the return value is also set to *cont* by default.
- Click on **OK** to close the interface dialog pane.

The names of the arguments and the return value for the method `doAddition` appear in the "Elements" pane on the left of the Block Diagram Editor. Now you can specify the functionality of the component by drawing a block diagram.

To specify the functionality of the component `Addition`:

- Drag the first argument from the "Elements" pane and drop it onto the drawing area of the block diagram editor.

The symbol for the argument appears in the drawing area.

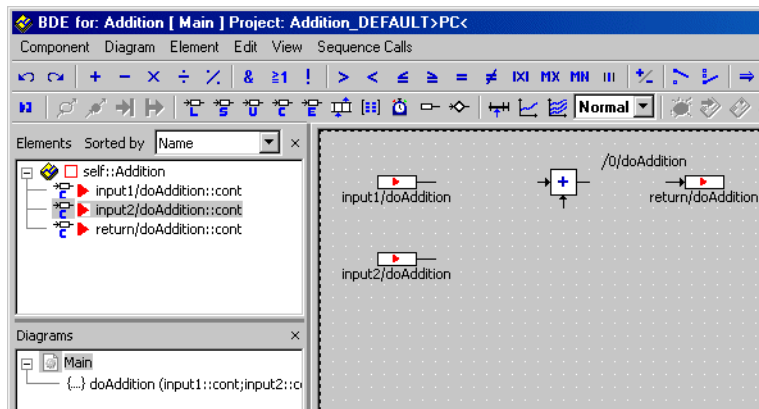


- Now add the other argument and the return value using the same drag-and-drop process.



- Click on the **Addition** button.
The mouse is loaded with an addition operator.
- Click inside the drawing area, between the symbols for the argument and for the return value.
An addition symbol is displayed. By default it has two input pins (indicated by arrows) and one output pin. The output pin is located on the right.

You can now arrange the elements and the operator by dragging them to their places on the drawing area. Next, you need to connect the elements to specify the flow of information.



To connect the diagram elements:

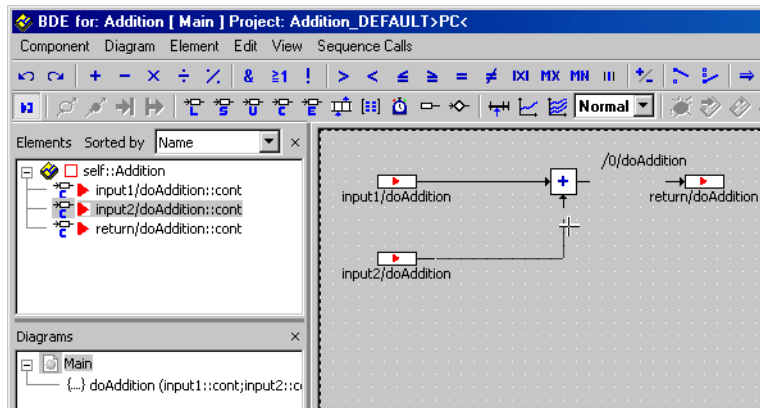


- Click on the **Connect** button.
- Alternatively, you can right-click in the drawing area (but not on an element).
The cursor changes to a crosshair when it is inside the drawing area.

- Click on the output pin of the first argument symbol to begin a connection.

Now, as you move the mouse cursor, a line is drawn after it. Every time you click inside the drawing area, the line remains fixed up to that point. That way you can determine the path of the connection line

- Click on the left input of the addition symbol.
- The argument symbol is now connected to the input of the addition symbol.

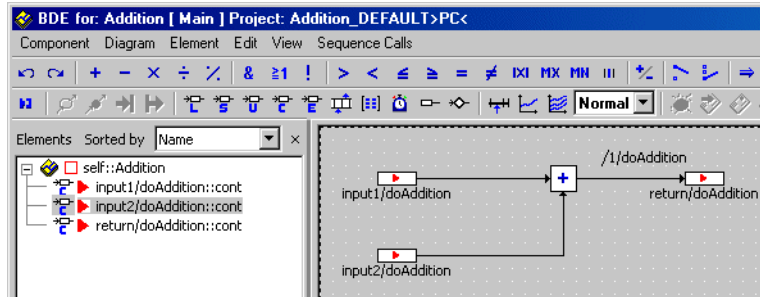


- Connect the second argument symbol with the other input of the addition symbol.
- Connect the return value symbol with the output of the addition symbol.

The connection between the addition operator and the return value is displayed as a green line to indicate that the sequencing for this operation needs to be determined.

- Select **Sequence Calls** → **Sequencing - Ignore Info** to determine the addition sequence automatically.

The connection between the addition operator and the return value is displayed as a black line.



Component specification is now complete. The last step in editing your component is to specify its layout, i.e., the way it is displayed when used within other components.

To edit the layout of a component:

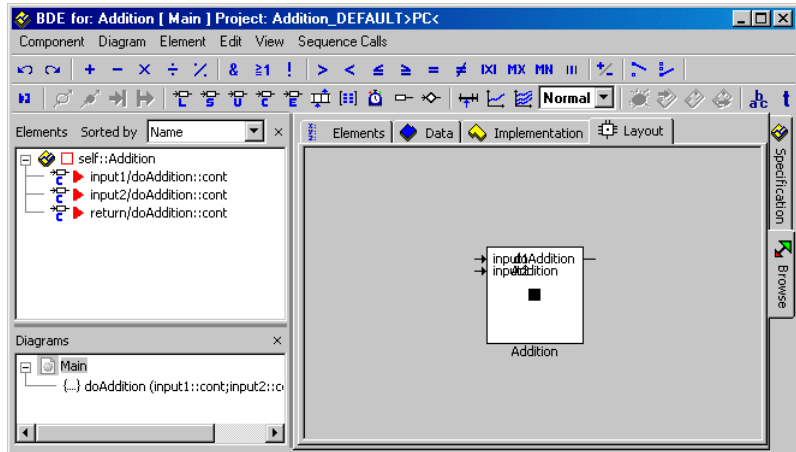
There are two ways to edit a layout:



- Use the **Browse** tab to go to the "Information/Browse" view.

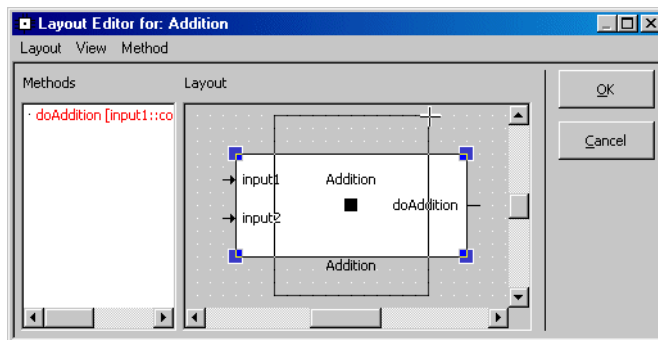
Name	Type	MaxSize	Scope	Kind	Existence	Dependency	Memory	Calibration	Unit	Comment
input1/doAddition	cont	---	doAddition	Method Argument	---	---	---	---		
input2/doAddition	cont	---	doAddition	Method Argument	---	---	---	---		
return/doAddition	cont	---	doAddition	Return Value	---	---	---	---		

- Double-click in the "Layout" tab to open the Layout Editor.



- Alternatively, select **Component** → **Edit Layout**.

The Layout Editor opens.



- Resize the block by clicking on it and then dragging the handles to the size you want.
- Drag the pins of the arguments and the return value to create a symmetrical design.
- Click on **OK**.

Now that you have finished your component, the last step in this lesson is to save the component in the database.

To save the component Addition:

- Select **Diagram** → **Store to Cache** and close the block diagram editor.
- In the Component Manager, select **File** → **Save Database**.

or



- click on the **Save** button.

Your work is not written to disk until you perform this operation.

When you select **Store to Cache**, the changes are only stored in the cache memory. It is therefore advisable to click **Save** regularly as work progresses.

You can have your changes saved automatically by activating the appropriate user options (see section "General Options" in the ASCET user's guide) for your ASCET session.

As an optional exercise, you could now model the same functionality in *ESDL* (*ESDL: Embedded Software Description Language*). If you continue with this exercise, you will familiarize yourself with the ESDL editor and will learn how to use the external source code editor.

The first step is to copy the module interface to a new module with type ESDL and rename it. Then create the functionality you want either directly in the ASCET ESDL editor or use the external text editor.

To copy and specify the component (interface) Addition:

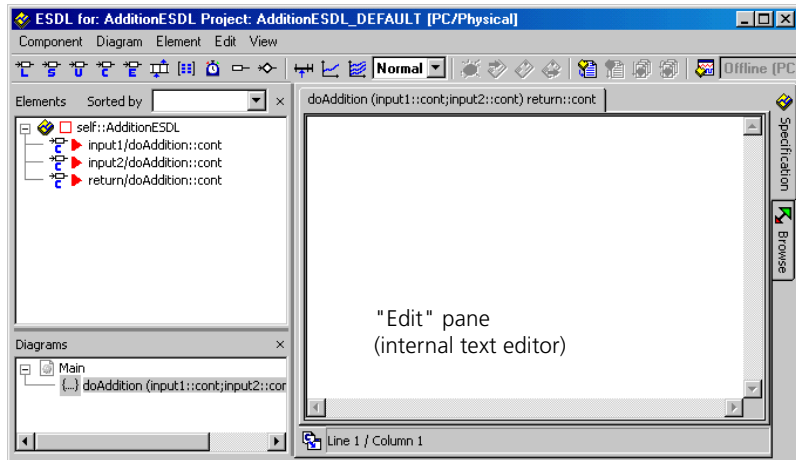
- In the "1 Database" pane of the Component Manager, select the component **Addition**.
- From the context menu select **Component** → **Reproduce As** → **ESDL**.

A copy of the component is created; it is named `Addition1`.

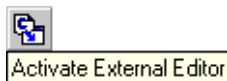
- Rename the new component `AdditionESDL`.

- In the "1 Database" list, double-click on the name of the new component.

The ESDL editor for `AdditionESDL` opens, making various functionalities available for editing.

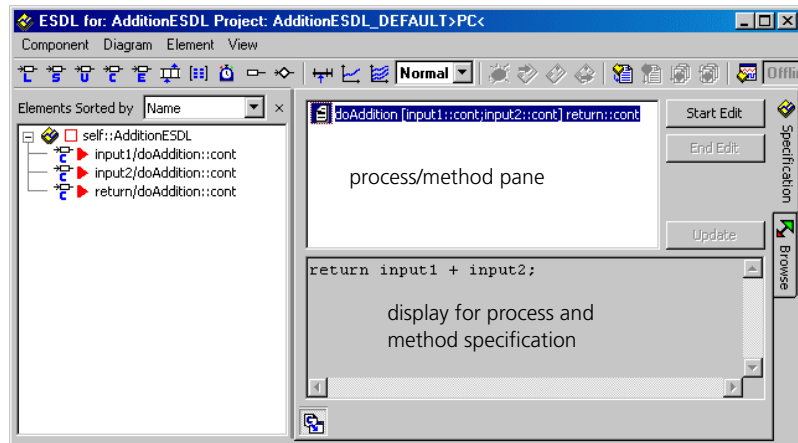


- Now enter this functionality in the "Edit" pane of the internal text editor:
`return input1 + input2;`
- Use the **Activate External Editor** button to switch to external editor mode.
 You are asked if you want to save your changes.
- Confirm with **Yes**.



The changes are saved, and the ESDL editor switches to "external editor" mode.

The editor looks different in "external editor" mode.



- In the process/method pane, select the method or process you want to specify.
The functionality entered previously appears in the specification field, and the **Start Edit** button is activated.
- Activate the external editor with **Start Edit**.

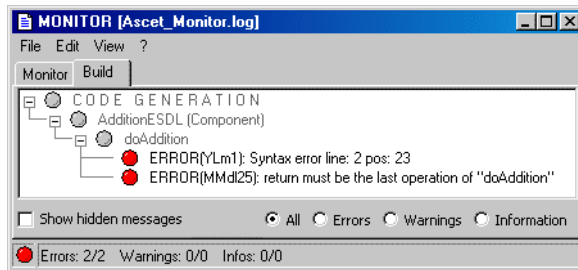
Note

*When the external editor starts up, the application associated with the file endings *.c and *.h in the operating system register database is called. Data transfer is done via temporary files; this is why you have to ensure the files are saved before closing the editor or before transferring to the ESDL editor.*

The **End Edit** and **Update** buttons are now activated, whilst the **Start Edit** button is deactivated.

- Edit the functionality in the external editor.
- Save the functionality in the external editor.
- In the ESDL editor, click **Update**, to transmit the data from the external editor.

- After you have finished, click **End Edit** to close the connection to the external editor.
The external editor remains open after you have clicked on **End Edit**. However, you cannot transmit any more changes to the ESDL editor.
- Click on **Activate External Editor** a second time to end the external editor mode.
- Select **Diagram** → **Analyze Diagram** to check the code you entered. Errors are listed in the ASCET monitor window.



6.1.3 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

- Opening a database
- Creating and naming a folder
- Creating and naming a component
- Defining the interface for a method
- Placing diagram elements on the drawing area
- Connecting diagram elements
- Editing the layout of a component
- Switching between Specification and Browser views.
- Saving a component.
- Copying a component interface.
- Using the ESDL editor.
- Using the external editor.

6.2 Experimenting with Components

Having created the `Addition` or `AdditionESDL` components, you can now experiment with them. Experimentation allows you to see how the component works, just as it would in a real application. The experimentation environment provides a variety of tools that can show the values of inputs, outputs, parameters and variables within a component.

6.2.1 Starting the Experimentation Environment

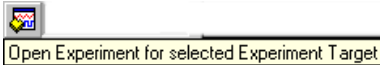
The experimentation environment is called from the block diagram or the ESDL editor. First open it with the component you want to experiment with.

To start the experimentation environment:

- From the ASCET Component Manager, open the block diagram editor for your `Addition` in the `\Tutorial\Lesson1` folder.
- In the block diagram editor, select **Component → Open Experiment**

or

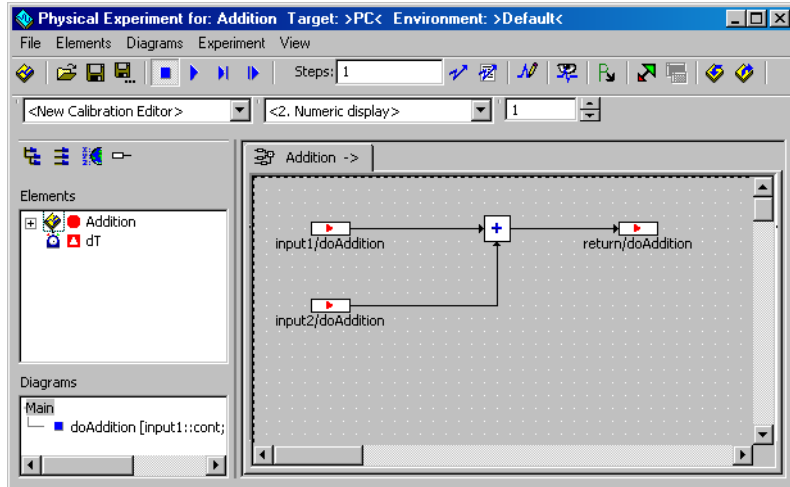
- click the **Open Experiment for selected Experiment Target** button.



The code for the experiment is generated. ASCET analyses the model in your specification and generates C code that implements the model. It is possible to generate specific code for different platforms.

In your example, you simply use the default settings to generate code for the PC.

After the code has been generated and compiled, the experimentation environment opens.



6.2.2 Setting up the Experimentation Environment

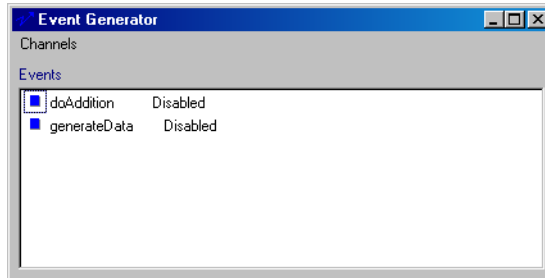
Before you can start experimenting, you have to set up the environment, which means determining the input values generated for the experiment and how you want to view the results. You have to carry out three steps. First set up the event generator, then the data generator and finally the measurement system.

To set up the Event Generator:



- Click on the **Open Event Generator** button. The "Event Generator" window opens. For each method to be simulated, you need to create an event and also a generateData

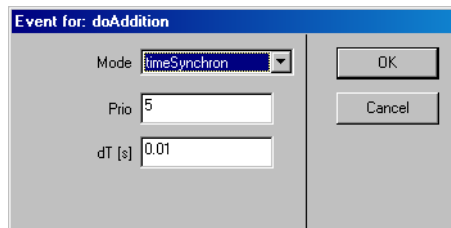
event. The events simulate the scheduling performed by the operating system of a real application.



- Select the event `doAddition`.
- Select **Channels** → **Enable**.
- Select the event `doAddition` again.
- Select **Channels** → **Edit**.

Both functions are also available via the context menu in the "Elements" field.

The "Event" dialog window opens.

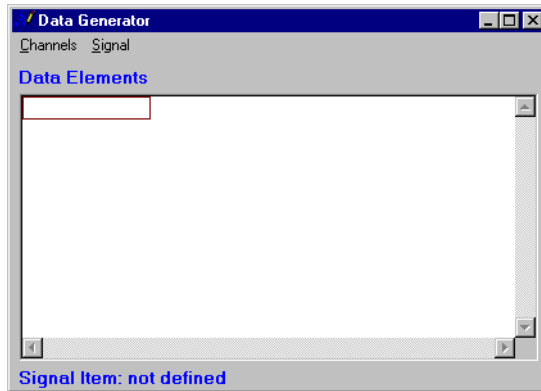


- Set the dT value to 0.001.
- Click on **OK**.
- In the event generator, select the `generateData` event and set its dT value to 0.001.
- Close the "Event Generator" window.

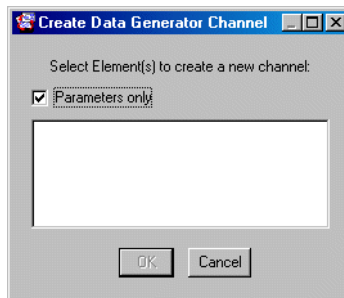
To set up the Data Generator:



- Click on the **Open Data Generator** button.
The "Data Generator" window opens.

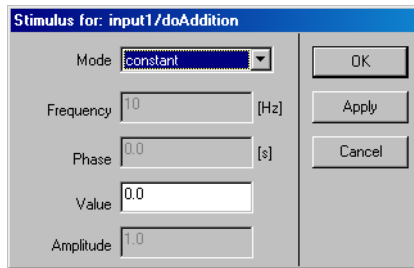


- Select **Channels** → **Create**.
The "Create Data Generator Channel" dialog window opens.



- If necessary, deactivate the **Parameters only** option to have both parameters and variables displayed in the list.
- Select the `input1/doAddition` and the `input2/doAddition` variables from the list.

- Click on **OK**.
Now both inputs are listed in the "Data Elements" pane of the "Data Generator" window.
- Select `input1/doAddition` in the "Data Elements" pane.
- Select **Channels** → **Edit**.
The "Stimulus" dialog box appears.



- Set the values as follows.

Mode:	sinus
Frequency:	1.0 Hz
Phase:	0.0 s
Offset:	0.0
Amplitude:	1.0
- Click on **OK** to close the "Stimulus" dialog pane.
- Set the values for `input2` as follows:

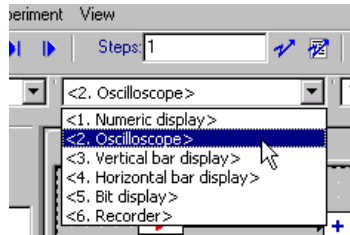
Mode:	sinus
Frequency:	2,0 Hz
Phase:	0.0 s
Offset:	0.0
Amplitude:	2.0
- Close the "Data Generator" window.

With these settings you get two sine waves with different frequencies and different amplitudes. The `Addition` component adds the two waves and displays the resulting curve.

In order to see the three curves displayed on an oscilloscope, you will now set up a measurement system.

To set up the measurement system:

- In the "Physical Experiment" window, select `<2. Oscilloscope>` as data display type from the "Measure View" combo box.

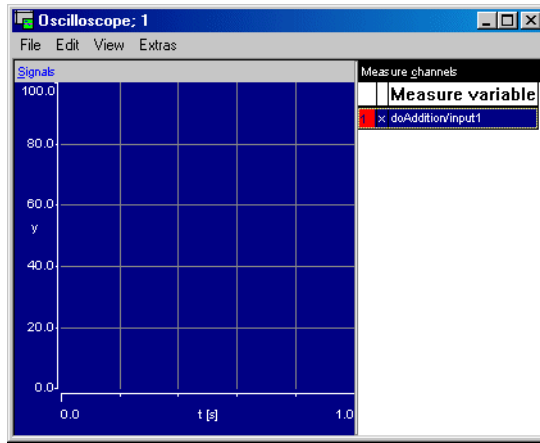


- In the "Elements" list, click on the box marked by a "+" next to `Addition` to expand the elements list.

The elements of the `Addition` component are displayed.

- Select `input1/doAddition`.
- Select **Elements** → **Measure**.

An oscilloscope window opens with `input1` as measurement channel. The "Measure view" list in the experimentation environment is updated to display the title of the measurement window.



- In the "Elements" list of the "Physical Experiment" window, select `input2/doAddition`.
- Select **Elements** → **Measure**.
`input2` is added to the oscilloscope as measure channel.
- Add `return/doAddition` to the measurement window.
- In the experimentation environment, select **File** → **Save Environment**.

Now the experimentation environment is set up, and you are ready to start the experiment. Since you have saved the experiment, it is automatically reloaded next time you start the experimentation environment for this component.

6.2.3 Using the Experimentation Environment

The experimentation environment provides a set of tools that allow you to view the values of all the variables in your component and also change the setup while the experiment is running. You can also adjust the way the values are displayed and choose from several ways of displaying them.

To start the experiment:



- In the "Physical Experiment" window, click on the **Start Offline Experiment** button.

The experiment starts running and the results are displayed in the oscilloscope.

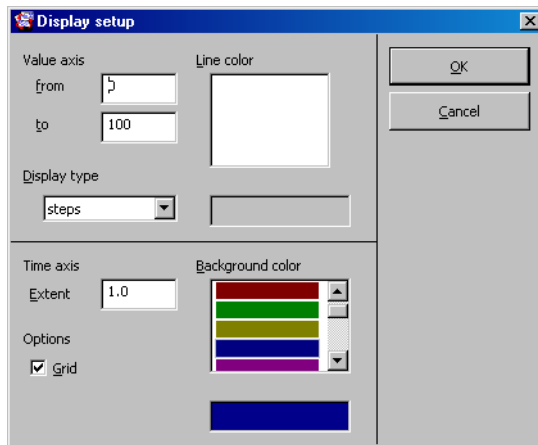


- Click the **Stop Offline Experiment** button to stop the experiment.

You will only see a small portion of the curves on the oscilloscope. To display the curves on the oscilloscope, you need to alter the scale on the value axis.

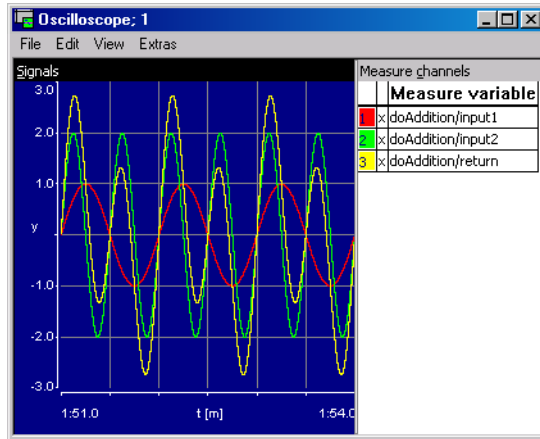
To change the scale on the oscilloscope:

- Select all three channels from the "Measure Channels" list in the oscilloscope window.
Hold the <CTRL> key while clicking on individual channels to select multiple items.
Now all the data elements are highlighted, so the changes you make will affect all three of them.
- Select **Extras** → **Setup**.
The "Display Setup" dialog box appears.



- Set the "Value Axis" to a range of -3 to 3.
- Set the "Time Axis Extent" to 3.
- Select a background color in the "Background color" list.

- Press <ENTER>.



The oscilloscope now shows the output with the appropriate scaling on the value axis. You will see the two input sine waves, together with the wave resulting from their addition. You can now adjust the input values to see how the output is affected.

To change the input values for experimentation:

- In the "Physical Experiment" window, select **Experiment** → **Data Generator** to open the "Data Generator" window.
- In the data generator, select the variable you want to change.
- Select **Channels** → **Edit**.
The "Stimulus" dialog box appears.
- Adjust the values you want to change.
- Click **Apply**.

The curves in the oscilloscope change according to the new settings. You can change all the settings in the experimentation environment while the experiment is running.

6.2.4 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

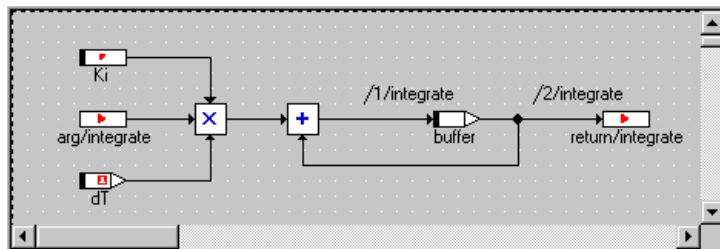
- To call the experimentation environment
- Setting up the event generator
- Setting up the data generator
- Setting up the measuring system
- Starting and stopping the experiment
- Saving the experiment
- Changing stimuli while the experiment is running

6.3 To Specify a Reusable Component

In this lesson you will create a class that implements an integrator, a standard piece of functionality that is often used in microcontroller software. While this is a slightly more complex diagram, the techniques for creating and experimenting with it are the same ones you have learned already.

In this example, you specify an integrator that calculates the distance covered where time and speed are known. The input value will be given in meters per second, and at each interval multiplied with a dT in seconds. The value for each time slice is added up in an accumulator. The accumulator stores the distance in meters that has been covered after a certain length of time.

In ASCET, a standard block, such as an accumulator, can be realized with a simple diagram.



6.3.1 Creating the Diagram

Before you start working on the diagram, you need to perform the same steps as for the `addition` component. First create a new folder in the `Tutorial` folder, then add a new class. Finally, you can specify the interface of the methods, then the block diagram and the layout.

You will start by creating the folder and the new class.

To create the integrator class:

- In the Component Manager, open the `Tutorial` folder.
- Create a new folder and call it `Lesson3`.
- In the `Lesson3` folder, create a new class and call it `Integrator`.

To define the integrator interface:

- In the "1 Database" pane, select the element `Integrator`.
- Double-click on the element or select **Component** → **Edit**.
The block diagram editor opens.
- Rename the method `calc` to `integrate` in the "Diagrams" pane.
- Edit the method `integrate` by adding one argument (type `cont`) and a return value (type `cont`).
- Click on **OK**.
The interface editor closes and you are back in the block diagram editor.
- Place the argument and return value from `integrate` on the drawing area.

The integrator uses two new types of elements that we have not used before: a variable and a parameter.

Variables are used in the same way as they are used in programming languages; you can store values in them and read the values for further calculations. In contrast, *parameters* are read-only. They can only be changed from outside, e.g. they can be calibrated in the experimentation environment, but they cannot be overwritten by any of the calculations within the component itself.

In addition, we want to specify a *dependent parameter* in this example. However, it is irrelevant for the functionality of the integrator. A *dependent parameter* is dependent on one or several parameters, i.e. its value is calculated based on a change in another one. The calculation or dependency is only carried out on specification, calibration or application. A dependent parameter behaves in exactly the same way in the target code as a normal parameter.

To create a variable:



- Click on the **Continuous** button.
The element editor opens.

Element Editor for: cont:cont

Name:

Unit:

Comment:

Dimension:

Model Type: Logic
 Signed Discrete
 Unsigned Discrete
 Continuous
 Enumeration

Kind: Constant
 System Constant
 Parameter
 Variable
 Input
 Output

Scope: Local
 Imported
 Exported

Existence: Virtual
 Non-Virtual

Dependency: Dependent
 Independent

Memory: Volatile
 Non-Volatile

Calibration: Yes
 No

Always show editor for new elements

Buttons: OK, Cancel, Formula

- In the "Name" field, enter the name `buffer`.
- Click **OK**.

The variable is now named `buffer`. The cursor shape changes to a crosshair. It is loaded with the continuous variable.

- Click inside the drawing area to place the variable.

The variable is placed in the drawing area. Its name is highlighted in the "Elements" list.

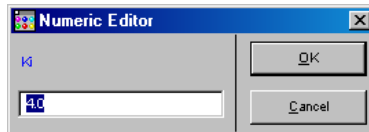
When the element editor does not open automatically, place the variable in the drawing area. Afterwards, double-click on the variable in the "Elements" list to open the element editor manually. Make the required settings and activate the **Always show dialog for new elements** option. The next time you create an element, the element editor opens automatically.

To create a parameter:



- Click on the **Continuous Parameter** button. The element editor opens.
- In the "Name" field, enter the name `κi`.
- Click **OK**.
- Click inside the drawing area to place the parameter.
- In the "Elements" list, right-click on the parameter and select **Edit Data** from the context menu.

A data configuration window (numeric editor) opens.



- Set the value in the window to 4.0 by typing it into the scalar calibration window and pressing <ENTER>.

This value becomes the default value for the parameter. You can assign default values to all parameters or variables in a diagram.

To create a dependent parameter:



- Click on the **Continuous Parameter** button.
The element editor opens.

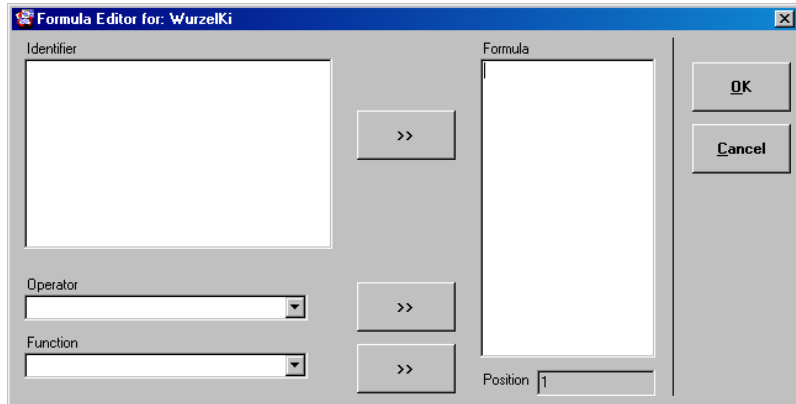
Element Editor for: cont_1::cont

Name	<input type="text" value="WurzelK"/>	<input type="button" value="OK"/>
Unit	<input type="text"/>	<input type="button" value="Cancel"/>
Comment	<input type="text"/>	
Dimension	Scalar	
Model Type	<input type="radio"/> Logic <input type="radio"/> Signed Discrete <input type="radio"/> Unsigned Discrete <input checked="" type="radio"/> Continuous <input type="radio"/> Enumeration	<input type="button" value="Formula"/>
Kind	<input type="radio"/> Constant <input type="radio"/> System Constant <input checked="" type="radio"/> Parameter <input type="radio"/> Variable <input type="radio"/> Input <input type="radio"/> Output	<input type="checkbox"/> Variants <input type="checkbox"/> Set() <input type="checkbox"/> Get()
Scope	<input checked="" type="radio"/> Local <input type="radio"/> Imported <input type="radio"/> Exported	
Existence	<input type="radio"/> Virtual <input checked="" type="radio"/> Non-Virtual	
Dependency	<input type="radio"/> Dependent <input checked="" type="radio"/> Independent	
Memory	<input type="radio"/> Volatile <input checked="" type="radio"/> Non-Volatile	
Calibration	<input checked="" type="radio"/> Yes <input type="radio"/> No	

Always show editor for new elements

- Name the parameter **WurzelKi**.
- In the "Dependency" field, activate the option **Dependent**.

- Open the formula editor using the **Formula** button.



- Right-click in the "Identifier" field and select **Add** from the context menu.
A formal parameter is created.
- In the "Identifier" field, enter the name x for the new parameter.
- In the "Formula" field, specify the calculation rule.

You can select different operators from the "Operator" combo box for more complex formulas. They are added to the "Formula" field one by one with the >> button next to the "Operator" field.

Likewise, various functions can be selected in the "Function" combo box, and inserted into the "Formula" field with the >> button next to the "Function" combo box.

- For the example here, select the root calculation of the formal parameter.

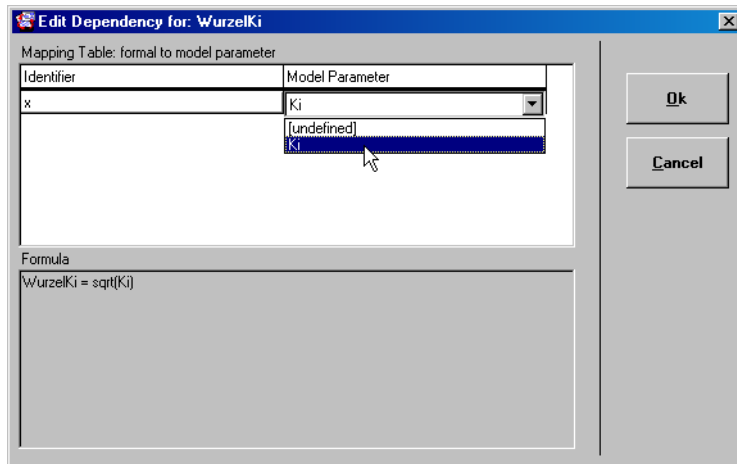
Formal Parameter: x

Formula: \sqrt{x}

- Exit with **OK**, and close the element editor, too.

The cursor shape changes to a crosshair.

- Click into the drawing area to place the parameter.
- In the block diagram editor, right-click on the `WurzelKi` parameter in the "Elements" list, and select **Edit Data** from the context menu.
- In the "Dependency Editor" window, assign a model parameter from the combo box to the formal parameter (in this example κ_i).

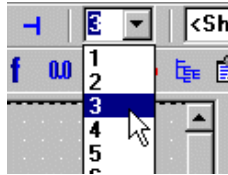


- Complete data entry with **OK**.
You have now specified a parameter dependent on the parameter κ_i which on calibration will automatically be calculated based on κ_i . Later on in the experiment, you can check the dependency or the calculation.

Now that you have added all the elements, you need to specify an integrator. You can proceed by creating the remainder of the diagram.

To create the diagram:

- In the "Argument Size" combo box, set the current value to 3 to specify the number of input values for the multiplication operator.



- Create a multiplication operator and place it on the drawing area.
- Click on the **dT** button to create a dT element. The element editor opens. All setting options are deactivated.

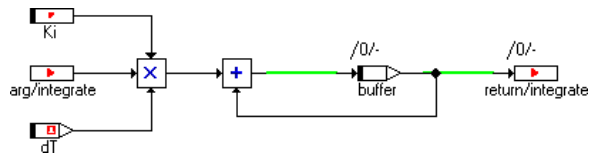


- Close the element editor with **OK**.
- Place the dT element inside the drawing area.
- Create an addition operator with *two* inputs and place it on the drawing area.

Be sure to set the argument size back to two before you create the operator.

- Connect the elements as shown below.

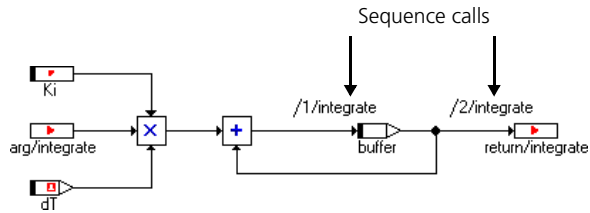
The input lines for both the buffer and the return value are displayed in green.



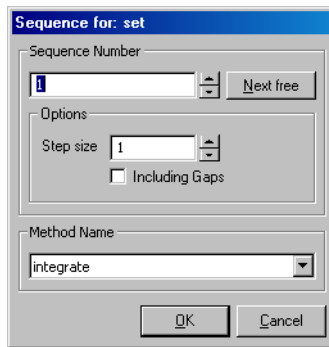
Now all the elements of the diagram are in place. Next, you need to determine the sequence of calculation by specifying the sequence calls.

To assign a value to a sequence call:

- Right-click on the sequence call above the variable `buffer`.



- Select **Edit** from the context menu.
The sequence editor appears.



- Click on **OK** to accept the default settings.
The assignment comes first in the algorithm for your integrator.

To adjust the sequence number in a sequence call:

- Right-click on the sequence call above the return value for `integrate`.
- Select **Edit** from the context menu.
- In the sequence editor, set the value for "Sequence Number" to 2.
- Click on **OK**.
The return value is assigned only after the variable `buffer` has been updated.

To adjust the layout:

- Select **Component** → **Edit Layout**.
The layout editor opens.
- Alternatively you can also get to the layout editor from the "Information/Browse" view by doubleclicking the layout within the "Layout" tab.
- Drag the argument from `integrate` to the middle of the left-hand side of the block.
- Drag the return value to the middle of the right-hand side of the block.
- Click on **OK**.

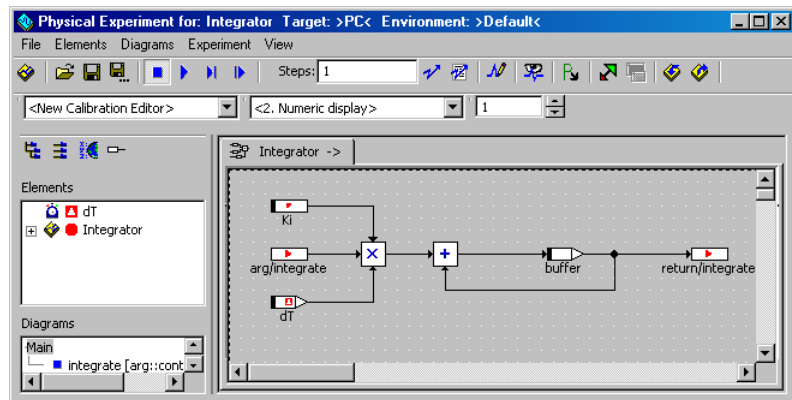
The diagram for the integrator class is now complete. Now save the changes to the diagram by selecting **Diagram** → **Store To Cache**. Changes that do not affect the diagram itself are stored automatically. Next save the changes to the database by selecting **File** → **Save Database** in the Component Manager window.

6.3.2 Experimenting with the Integrator

Again, first set up the event generator, then the data generator and finally the measurement system.

To set up the experimentation environment for the integrator:

- Start the experimentation environment by selecting **Component** → **Offline Experiment**.





- Click the **Event Generator** button.
- Activate the event `integrate` using the default `dt` value of 0.01.
- Close the "Event generator" window.



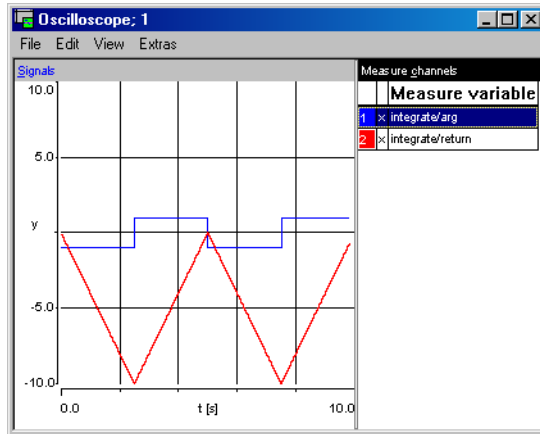
- Click on the **Data Generator** button.
- Create a data channel for the `integrate` method by selecting **Channels** → **Create** and selecting the argument from `integrate`.

- Set the values as follows:

Mode: pulse
Frequency: 0,2 Hz
Phase: 0.0 s
Offset: -1.0
Amplitude: 2.0

- Close the Data Generator.
- Open an oscilloscope window with the `arg` and `return` values from the `integrate` method.
- Set the value axis to a range from -10 to 10 and the time axis extent to 10 seconds.
- Select **Start Offline Experiment** to start the experiment.

The output value of the `integrate` method increases when the argument is positive, and decreases when it is negative. Because the positive and negative parts of the input curve are equal, the output will remain within stable boundaries.



To reset an experiment:

If you stop an experiment and then restart it again, all the variable values will be stored. Sometimes it may be desirable to reset all the variables to their initialization values.

- Select **Elements** → **Reinitialize All** → **Variables** or **Parameters** or **Both**.

Depending on your selection, either all variables or all parameters, or both, are reset to their initialization values.

Next, you should experiment with various settings to illustrate the function of the integrator. You can adjust the κ_i parameter and change the input.

To experiment with the integrator:

- In the "Elements" pane, click on the plus sign next to **Integrator** to expand the **Integrator** element.

- Select the parameter κ_i .

- Select **Elements** → **Calibrate**.

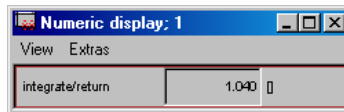
A numerical editor opens for the parameter.

- Set the value to 5 by entering the value and then pressing <ENTER>. The output curve on the oscilloscope becomes steeper.
- Set the value to 3. The output curve now becomes flatter again.
- Set the parameter back to 4 and close the numeric editor.
- Open the "Data Generator" window.
- Set the offset of the input pulse to -0.5.
- Click on **OK**.

Now the positive part is greater, so the output will start to increase. At some point it will exceed the oscilloscope limits. You can adjust the scale of the oscilloscope for each value individually by selecting only that value when you make changes. You can also open a numerical display window to see the output value.

To display a value numerically:

- Select <1.Numeric Display> in the "Measure View" combo box in the experimentation environment.
- In the "Elements" pane, select the return value (return) from the integrate method.
- Select **Elements** → **Measure**.
A "Numeric display" window shows the current return value.



- Also display the dependent parameter wurzelki.
- Experiment with changing κ_i and initiating update using the **Update Dependent Parameters** button.



6.3.3 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

- Creating a parameter
- Creating and specifying a dependent parameter
- Creating a variable:
- Creating an operator with multiple inputs
- Setting the sequence number of a sequence call
- Assigning a default value
- Calibrating a value during experimentation
- Displaying values in a "Numeric display" window

6.4 A Practical Example

In this lesson you will create a controller based on a slightly enhanced standard PI filter. The controller will be used to keep the rotational speed of an idling car engine constant.

When controlling the idling speed of an engine, you have to make sure that the actual number of revolutions n stays close to the nominal value for idling n_{nominal} . The value n is subtracted from n_{nominal} to determine the deviation that is to be controlled.

The deviation in the actual number of revolution forms the basis for calculating the value of $\text{air}_{\text{nominal}}$, which determines the throttle position, i.e. the amount of air the engine gets.

6.4.1 Specifying the controller

The steps in creating the diagram for your controller are the same as earlier:

- adding a new folder and creating the component in the Component Manager,
- defining the interface and drawing the block diagram.

The major difference is that you will implement the controller as a module. Modules are used as the top-level components in projects. They define the processes that make up a project.

To create the controller component:

- In the Component Manager, add a new subfolder to the Tutorial folder and rename it Lesson4.
- Select the Lesson4 folder and select **Insert** → **Module** → **Block diagram** to add a new module.
- Rename the new module IdleCon and open the block diagram editor.
- In the "Diagrams" pane, rename the diagram process to p_idle.

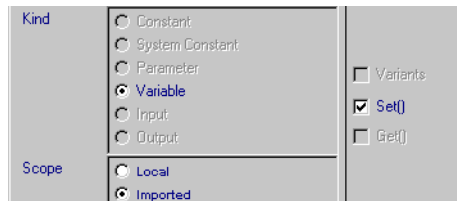
The functionality of modules is specified in processes, which correspond to the methods in classes. Unlike methods, processes do not have arguments or return values. Data exchange (communication) between processes is based on directed messages, which are referred to as *Receive messages* (inputs) and *Send messages* (outputs) in ASCET.

In your controller, you will use a receive message to process the actual number of revolutions n and a send message to adjust the throttle position to `air_nominal`.

To specify the interface of the controller:

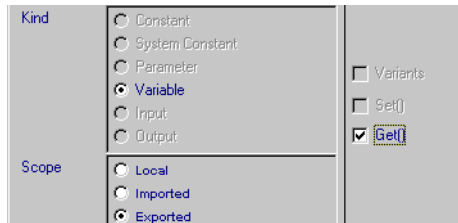


- Create a receive message by clicking on the **Receive Message** button and name it n .
- In the element editor for the message n , tick the **Set()** option.



- Click on the **Send Message** button and then inside the drawing area to create a send message.
- Rename it `air_nominal`.

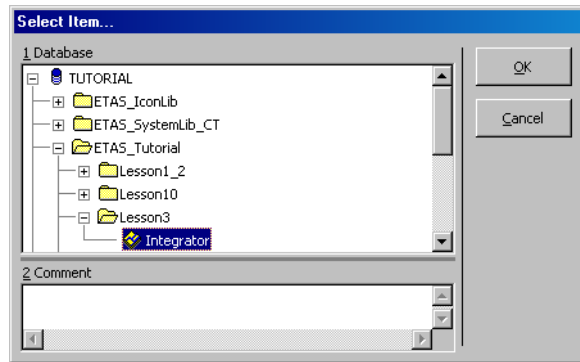
- In the element editor for the message `air_nominal`, tick the **Get()** option.



The controller element uses the integrator you created in Lesson 3.

To add the Integrator to the controller:

- Select **Element** → **Add Item** to open the "Select item" dialog box.



- In the "1 Database" pane, select the item `Integrator` from the `Tutorial\Lesson3` folder and click **OK** to add the integrator.

The integrator is included in the component `Id1eCon`. A component is included by reference, i.e., if you change the original specification of the integrator, that change will be reflected in the included component.

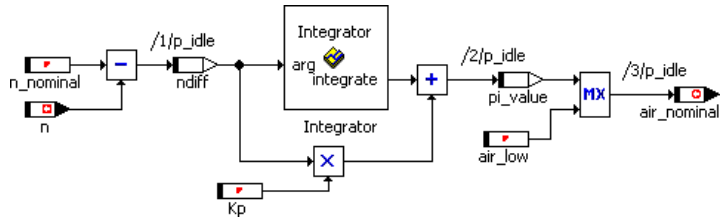
In addition to the elements you have added so far, you need to add the following elements to your controller:

- two continuous variables, named `ndiff` and `pi_value`

- three continuous parameters named `n_nominal`, `Kp`, and `air_low`

To specify the remainder of the controller:

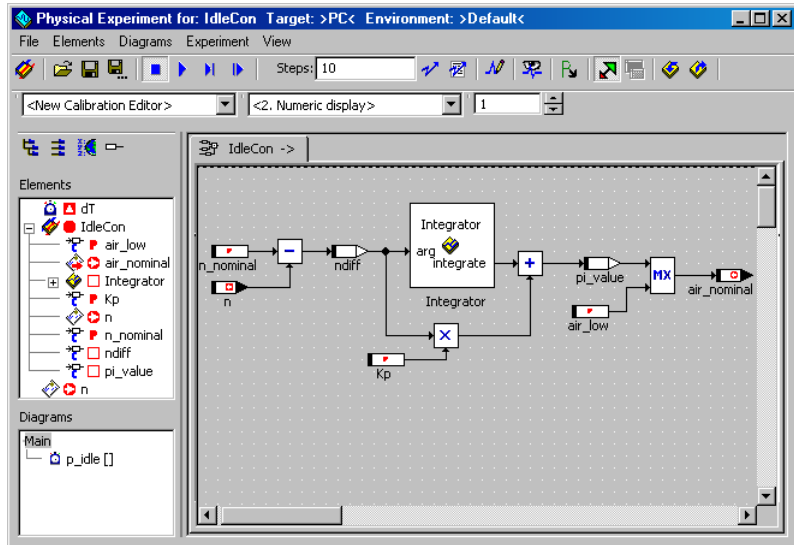
- Create the operators and the other elements needed, then connect them as shown in the block diagram below.



- In the "Elements" list, select the `n_nominal` parameter, then select **Element** → **Edit Data**.
- Set the value for `n_nominal` to 900.
- Set the value for `Kp` to 0.5.
- Save your specification in the diagram and apply the changes to the database.

6.4.2 Experimenting with the Controller

Experimentation with modules works like experimentation with other components. First the data and event generators and then the measurement system are set up.



To set up the experimentation environment:

- Select **Component** → **Open Experiment** to start the experimentation environment.
- Open the "Event Generator" window and enable the event for the process `p_idle` using the default value of 0.01 for `dT`.
An event for a process works the same as an event for a method.
- Open the "Data generator" window and set up the channel for the receive message `n` with the following values:

Mode: pulse
Frequency: 1.0 Hz

Phase: 0.0
Offset: 800.0
Amplitude: 200.0

- Set up an oscilloscope with the variables `n_diff` and `air_nominal`.
- In the oscilloscope, set the value axis to -500 to 500 and the time axis extent to 2.
- Click on the **Save Environment** button.



The experiment is now set up to display the relationship between the deviation in the number of revolutions and the throttle position.

To experiment with the controller:



- Start the experiment by clicking the **Start Offline Experiment** button.
- Open a calibration window for the variables κ_i and κ_p . From here, you can adjust the values κ_i and κ_p and observe their effect on the output.

From time to time, you may need to reinitialize the model in order to get back to meaningful values.

6.4.3 A Project

A project is the main unit of ASCET software representing a complete software system. This software system can be executed on experimental or microcontroller targets in real-time with an online experiment. Individual components can only be tested in the offline experimentation environment.

Every experiment runs in the context of a project. Whenever code is generated for a project, the operating system code is also generated. The operating system specification is required to run on an ASCET software system in real-time. Running a software system in real-time is called *Online experimentation*. So far, we have experimented *offline* only, i.e. not in real-time.

Note

All ASCET experiments—both online and offline—run within the context of a project. This is clearly seen with offline experiments, which use an (otherwise invisible) default project. Creating and setting up a project for the express purpose of specifying an operating system is only required for online experiments. However, you also have the option of configuring the default project for your own application.

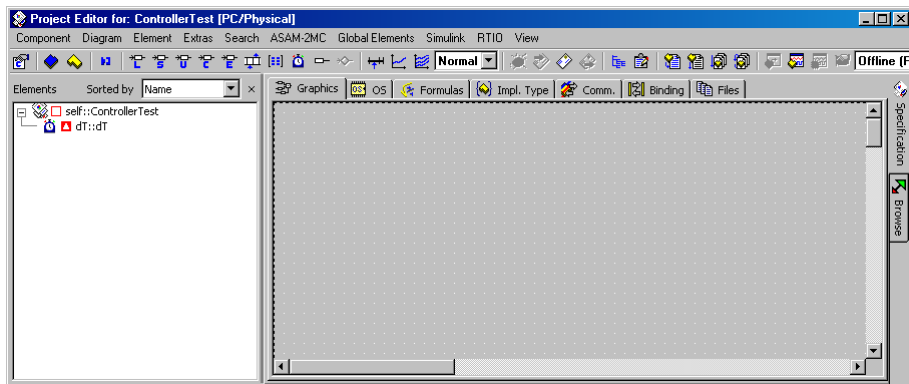
6.4.4 To set up the Project

The project is created in the Component Manager. You can add it to the same folder as the IdleCon module.

To create a project:



- In the Component Manager, select **Insert** → **Project** or click on **Insert Project** to add a new project.
- Name the project **ControllerTest**.
- Select **Component** → **Edit** or double-click the project.
The project editor opens for the project.



The next step is to add the `IdleCon` controller to the "Elements" list of the project.

To include components in a project:

- In the project editor, select **Element** → **Add Item** to open the "Select item" dialog box.
- From the "1 Database" list, select the component `IdleCon` in the `Tutorial\Lesson4` folder.
- Click on **OK** to add the component.
The name of the component is shown in the "Elements" list of the project editor.

Components are included by reference, i.e. if you change the diagram of an included component, that change will also be effective in the project.

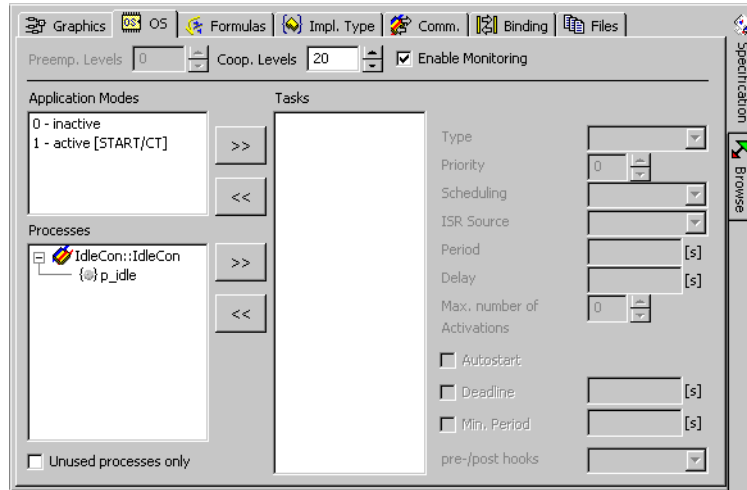
The operating system schedules the tasks and processes of a project. Before you can generate code for the project, you have to create the necessary tasks and assign the processes to them.

The operating system schedule is specified in the "OS" tab of the project editor. You will now specify the operating system schedule to have the `p_idle` process activated every 10 ms.

To set up the operating system schedule for the project:



- Click on the "OS" tab.



- Select **Task** → **Add** to create a new task.
- Name it `Task10ms`.

Newly created tasks are by default alarm tasks, i.e. they are periodically activated by the operating system.

- Assign the task a period of 0.01 seconds in the "Period" field.

The period determines how often the task is activated, which is every 10 ms in this case.

- In the "Processes" list, expand the `IdleControl` item by clicking on the plus sign next to it.



- Select the process `p_idle` and select **Process** → **Assign**.

The process is assigned to the `Task10ms` task. It is displayed beneath the task name in the "Tasks" list.

In projects, imported and exported elements are used for inter-process communication. They are global elements that correspond to the send and receive messages in the modules. Global elements must be declared in the project and linked to their respective counterparts in the modules included in the project.

To define global elements:

- In the project editor, select **Global Elements** → **Resolve Globals**.

The necessary global elements are created and automatically linked to their counterparts. Elements with the same name are automatically linked to each other.

Note here that send messages are defined in the module (exported) by default.

6.4.5 Experimenting with the Project

You will now run an offline experiment with this project. Offline experimentation can be performed on the PC without the connection of any additional hardware. Projects run on the PC by default. Therefore you do not have to adjust any settings. Offline experimentation with projects works like offline experimentation with components.

To set up the experimentation environment:

- In the Component Manager, select **File** → **Save Database**.

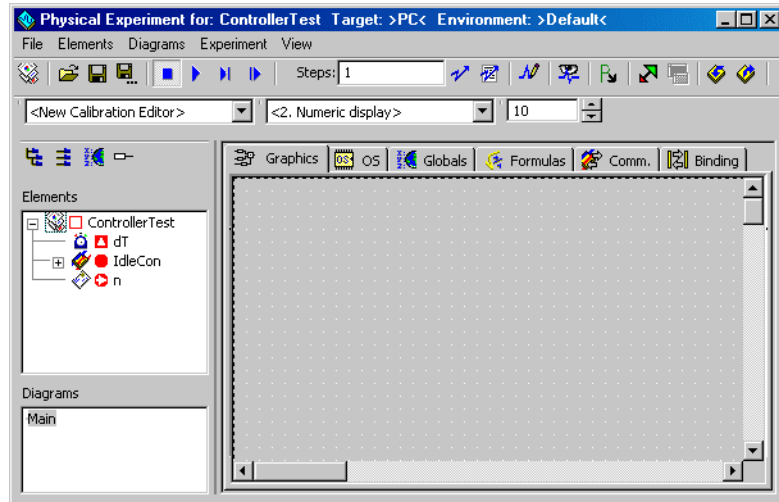
It is always a good idea to apply your changes to the database before you start the experimentation environment.



Open Experiment for selected Experiment Target

- Click on the **Open Experiment for selected Experiment Target** button.

Code for the project is generated and the offline experimentation environment opens.



- Click on the **Open Event Generator** button. In the event generator you see an event for each task you want to use in the experiment, rather than for each method or process, as in experimentation with components.
- Enable the task `generateData` from the event generator and use the default `dT` value of 0.01 seconds.

The task `Task10ms` is already enabled by default, and both events now have 0.01 seconds as their `dT` value; therefore you do not need to make any further adjustments.

- Close the event generator.
- Set up the data generator and measurement system with the same values as in the previous experiment (cf. "Experimenting with the Controller" on page 141).

- Save the environment by selecting **File** → **Save Environment**.

To run the experiment:



- Click on the **Start Offline Experiment** button.
- Adjust the κ_i and κ_p parameters as in the previous section to see the effect of your changes in the output.

6.4.6 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

- Creating modules
- Creating messages in modules
- Using components from the Component Manager in a block diagram.
- Creating a project
- Including components in projects.
- Creating tasks and assigning processes to them
- Experimenting with projects

6.5 Extending the Project

In this lesson you will add some refinements to make your controller more realistic. You will create a signal converter that converts sensor readings into actual values. Many sensors, used for instance in automotive applications, return a voltage that corresponds to a particular measurement value, such as temperature, position or number of revolutions per minute. The relationship between the voltage and the measured value is not always linear. ASCET provides characteristic tables to model this kind of behavior efficiently.

6.5.1 Specifying the Signal Converter

The first step in modeling the signal converter is to create a folder and a module that specifies the functionality. The signal converter uses two characteristic lines to map its input values to the corresponding outputs.

To create the module:

- In the Component Manager, create a new folder `Tutorial\Lesson5`.

- Create a new module and name it `SignalConv`.
- Select **Component** → **Edit** or double-click the element to open the block diagram editor.
- In the block diagram editor, select **Diagram** → **Add Process** to create a second process.
- Name the processes `n_sampling` and `t_sampling`.
- In the "Elements" list, create two receive messages `u_n` and `u_t` and two send messages `t` and `n`.

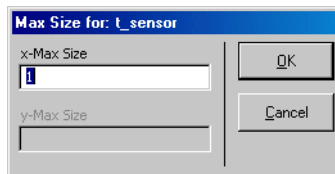


- Create a characteristic field by clicking on the **One-D Table Parameter** button.

The element editor opens.

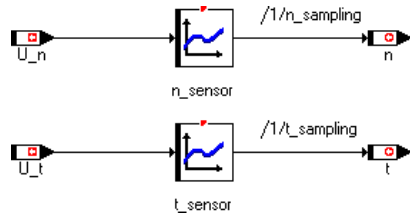
- Call the table `t_sensor` and close the element editor.

The "Max Size" dialog window opens. Since you created a one-dimensional characteristic line, the "y-Max Size" field is deactivated.



- In the "x-Max Size" field, enter the value 13.
The characteristic field can now span a maximum of 13 columns.
As you have created a one-dimensional characteristic line, the "y-Max Size" field is inactive.
- Click **OK** to close the dialog box.
- Then click in the drawing area to place the table.
The table is added to the "Elements" list.
- Create a second table with a maximum of 2 columns and call it `n_sensor`.

- Connect the elements as shown and edit the sequencing to assign the corresponding processes.



The next step is to edit the data for the two characteristic fields. ASCET provides a table editor for editing arrays, matrices and characteristic fields.

To edit the tables:

- Right-click on the table `t_sensor` and select **Edit Data** from the context menu.

The table editor opens.

- Adjust the size of the table as follows:



The table is extended to 13 columns with all z-values set to 0 by default.

- Enter the values listed in the following table. The top row corresponds to the X row, the bottom row to the Z row.

0.00	0.08	0.30	0.67	1.17	2.5	5.00	7.50	8.83	9.33	9.70	9.92	10.00
-40.0	-26.0	-13.0	0.0	13.0	40.0	80.0	120.0	146.0	160.0	173.0	186.0	200.0

You should edit the table by entering the sample points (X values) first, starting from left to right.

- Click on an X value and then enter the new one in the dialog box.

The new X value must be between the limits set left and right by the sample points.

- Then enter the output values by clicking on a value and typing over the highlighted value.
- Edit the second table in the same way using the following data:

0.0	10.0
0.0	6000.0

- In the block diagram editor, select **Diagram** → **Store to Cache**.
- In the Component Manager, click on the **Save** button to store your changes.

In this example, the second table represents a linear relationship between input and output, therefore it needs only two sample points. This works because you have specified the interpolation mode between values as linear.

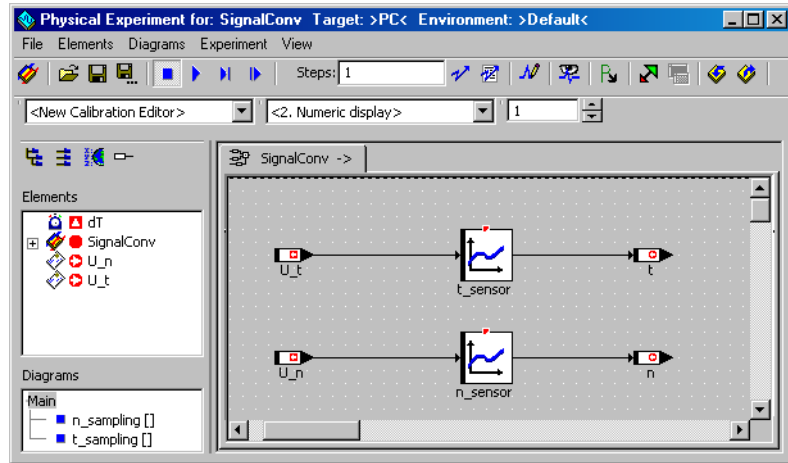
In linear interpolation, for an input value between two sample points the output value is determined from a straight line. In this case, an input of 0 returns 0 and an input of 10 returns 6000. If the input value is 5, the return value is interpolated accordingly as 3000.

6.5.2 Experimenting with the Signal Converter

You can now experiment with the new component to observe the behavior of the tables. Since the two tables have different value ranges, you will set up a separate oscilloscope window for each of them.

To set up the experimentation environment:

- Select **Component** → **Open Experiment** to open the experimentation environment.

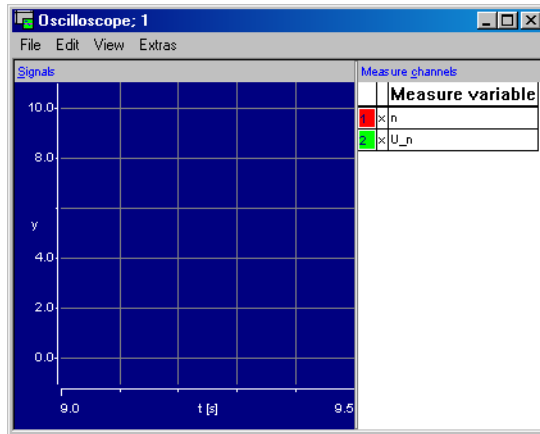


- Create an event for each process in the component (`n_sampling`, `t_sampling`, `generateData`) and assign a `dT` value of 4 ms to each event.
- In the data generator create a channel for the message `U_n` and one for `U_t` and set up both channels with the following values:

Mode: sinus
Frequency: 2,0 Hz
Phase: 0.0
Offset: 5.0
Amplitude: 5.0

- Create an oscilloscope window with the messages n and \bar{u}_n and a second oscilloscope with the messages t and \bar{u}_t .

Before you create the second oscilloscope, be sure to activate the `<2. Oscilloscope>` entry in the "Select Measure View" combo box.



The resolution of the sampling points and their corresponding interpolation values differs so much that you should configure each channel in the two oscilloscopes individually in order to optimize the way the behavior of the two tables is displayed.

To set up the oscilloscopes for measuring:

- Activate the oscilloscope for the process `n_sampling` (channels \bar{u}_n and n).
- In the "Measure Channels" list, select the message n and select **Extras** → **Setup**.
The "Display Setup" dialog box for the message n is displayed.
- Set the range of the value axis to 0 to 6000 and the time axis to 0.5
- Open the "Display Setup" dialog box for the message \bar{u}_n .

- Set its value axis to a range from -1 to 11.
The time axis must be the same for all variables in an oscilloscope window, so you do not have to change that.
- Activate the oscilloscope for the process `t_sampling` (channels `U_t` and `t`) and set up its channels as follows:

	<code>U_t</code>	<code>t</code>
Min	-1	-40
Max	11	200
Extent	0.5	0.5



- Save the experimentation environment by clicking the **Save Environment** button.

You are now ready to run the experiment and see how your signal converter works. Observe the differences between the two conversion modes.

To run the experiment:



- Click on the **Start Offline Experiment** button.

In the `n_sensor` table, only the amplitude of the input sine wave changes. The input here is a voltage signal ranging from 0 to 10 volts, this is mapped to the rotational speed, ranging from 0 to 6000 revolutions per minute. The table `t_sensor` does not represent a linear relationship between the input voltage and the output temperature. It matches the characteristic behavior of temperature sensors commonly used in the automotive industry.
- Change the data generator channels to different wave-forms and observe the effect on both output curves.

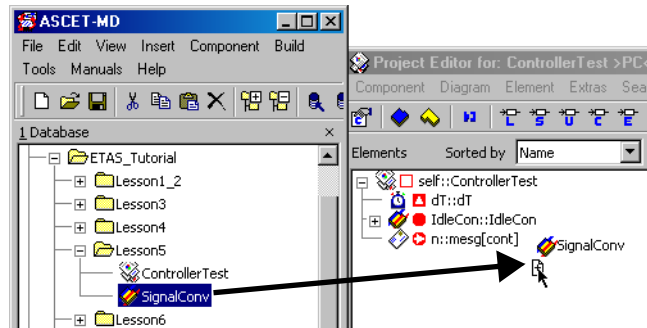
6.5.3 Integrating the Signal Converter into the Project

After you have specified the signal converter, you can integrate it in the project you created in Lesson 4. The output signal for the signal converter is used as the input signal for the motor controller.

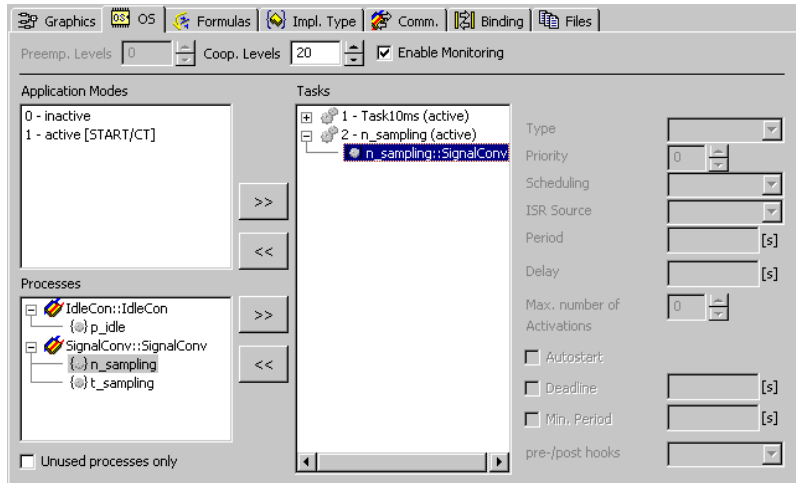
To integrate the signal converter in the project, you will set up another task in the operating system schedule for the new processes and declare and link the global elements necessary for the processes to communicate.

To add the signal converter to the project:

- From the Component Manager, open the project editor for the project `ControllerTest`.
- Drag the module `SignalConv` from the "1 Database" list of the Component Manager to the "Elements" list of the project.



- Click on the "OS" tab to activate the operating system editor.
- Create a new task `n_sampling`.
- Set the period for the new task to 0.004 seconds.
- Assign the process `n_sampling` to the task `n_sampling`.



The project now has two tasks. The first task is activated every 10 milliseconds, the second one every 4 milliseconds. All the processes assigned to a given task are executed at the interval specified. In the example, each task has only one process, but it is possible to have any number of processes per task.

The next step in integrating the signal converter is to resolve communication between the modules. Communication between the processes works through global elements. All global elements used within a project have to be defined as messages in the corresponding modules.

By default, send messages are defined in a module while receive messages are normally only imported into a module so they have to be defined now within the context of the project.

To set up the global elements:

- Select **Global Elements** → **Resolve Globals** to set up automatic links.
- Select **Global Elements** → **Delete Unused Globals** to remove the links from the previous lesson.

All the necessary global elements are created and linked automatically to the corresponding elements with a matching name. The global message \cup_n , for instance, is automatically linked to the message \cup_n in `SignalConv`.

Note that it is necessary to delete unused globals because the message `n` was defined in lesson 4 in the project context while it is now defined in the module `SignalConv`. The message `n` is now used for communication between the processes of the modules.

To experiment with the project:



Open Experiment for selected Experiment Target

- Click on the **Open Experiment for selected Experiment Target** button to activate the experimentation environment.
- Open the event generator and enable the task `n_sampling`.
- Set the `dT` value for the task to 4 milliseconds. During offline experimentation with projects, the event generator simulates the scheduling that is performed by the operating system during online experimentation.
- Open the data generator and delete the existing data channel.
- Then set up a new channel for the message `U_n`.
- Set up the channel `U_n` as follows:

Mode:	pulse
Frequency:	1.0 Hz
Phase:	0.0
Offset:	4/3
Amplitude:	1/3

- Now activate `U_n`, the output voltage of the rotational speed sensor.

The signal converter converts the voltage value into the actual value for `n` using the characteristic table `n_sensor`.

The values given above produce an output range for `n` that matches the range from the previous experiment (without signal processing).

- Click on the **Save Environment** button.
- Start the experiment.



Save Environment

The output curves should be the same as in the example without signal processing. The stimulus created by the data generator is different, but is then processed in the table so that it looks the same as before.

6.5.4 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

- Creating and using characteristic fields
- Adding components to a project
- Define the communication between different components in a project

6.6 Modeling a Continuous Time System

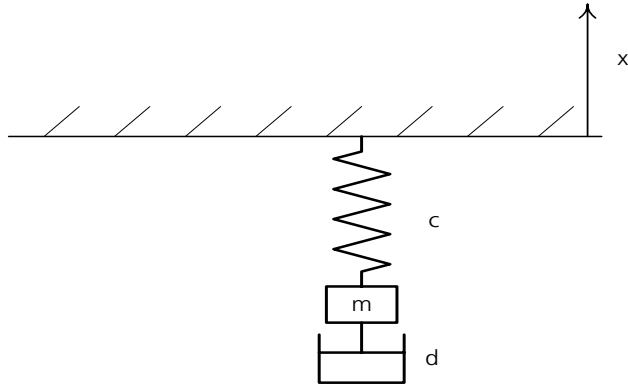
The realistic modeling of physical, mechanical, electrical, and mechatronic processes, often described by differential equations, requires continuous time methods. Before integrating a method like this in the project created in the previous chapters, this chapter covers modeling a time continuous system using a detailed example.

ASCET supports the modeling and simulation of continuous time systems by means of so-called CT blocks. CT stands for "Continuous Time" and refers to items that are modeled or calculated in quasi-continuous time intervals. The continuous time modeling in ASCET is based on state space representation, the standard description form used in the design of continuous time systems. This representation allows the description of CT basic blocks by nonlinear ordinary first-order differential equations and nonlinear output equations. ASCET provides several real-time integration methods to find optimal solutions to these differential equations (refer to chapter 8.2 "Solving Differential Equations – Integration Algorithms" of the ASCET reference guide).

The procedure for modeling a continuous time system will now be explained using the example of a mass-spring pendulum with attenuation by the earth's gravity.

6.6.1 Motion Equation

The mass m shown in the following illustration is subject to the following forces:



- gravity: $F_g = -mg$
(g = gravitational acceleration)
- Spring force: $F_F = -c(x + l_0)$
(c = spring rate, l_0 = length of spring at rest, and x = position of mass m)
- Attenuation $F_D = -d \dot{x}$
(d = attenuation constant and \dot{x} = velocity of mass)

This gives the motion equation as follows:

$$m\ddot{x} = -mg + F \text{ Or } \ddot{x} = -g + F/m \text{ (with } F = F_F + F_D)$$

Breaking the second-order differential equation into two first-order differential equations ($\dot{x} = v$, $\dot{v} = \ddot{x}$) results in:

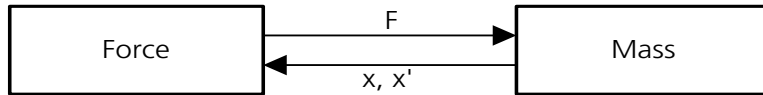
$$\dot{x} = v$$

$$\dot{v} = -g + F/m$$

These differential equations will be used in the following model design.

6.6.2 Model Design

For simplicity, the model of the mass-spring pendulum will be designed using a single CT block. However, to illustrate the "direct pass-through" or "non-direct pass-through" properties and to demonstrate how to avoid an algebraic loop by skillful setting of these properties, we will design this model using two blocks.



- The `Force` block calculates spring force F from the position of the pendulum's mass m and the friction force from the velocity x' .
- From the spring force F the `Mass` block calculates the acceleration x'' from the integration of which the velocity x' and the position x result.

At first sight, this system looks like an algebraic loop: each block expects an input value from the other block in order to calculate an output value required by the other block.

This algebraic loop can be avoided by clever setting of the *direct pass-through* or *non-direct pass-through* properties:

- In the `Force` block, the output variable F via the equation

$$F = -c(x + l_0) - dx'$$

is directly dependent on the input variables x and x' . This block is thus defined as having a direct pass-through.

- In the `Mass` block however, the output variables x and x' do not depend directly on the input variable F , but on the internal state variables of the block. These, at least at the start, have initial values from which the output variables x and x' can be calculated, when the input variable F is unknown. Otherwise the output variables are calculated using the following differential equations:

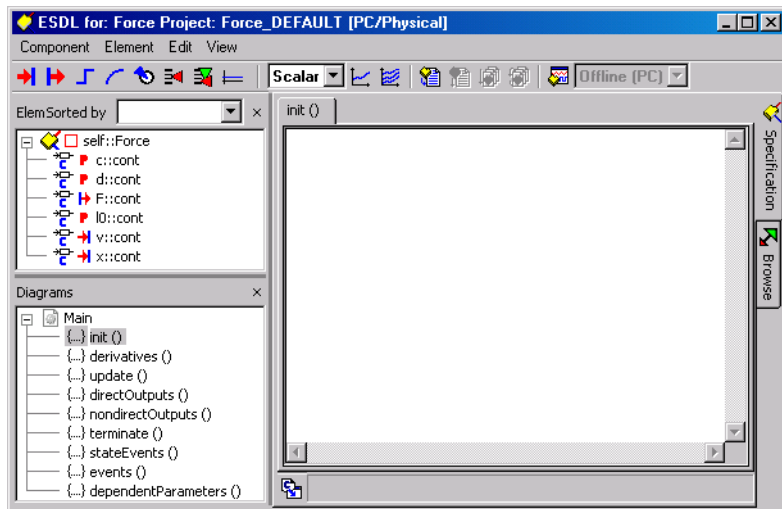
$$\begin{aligned}x' &= v \\v' &= -g + F/m\end{aligned}$$

This block is thus defined as having a non-direct pass-through.

Model Creation

- In the Component Manager, create a folder with **Insert** → **Folder** and call it `Lesson6`.

- In this folder, use **Insert** → **Continuous Time Block** → **ESDL** to create a block `Force` and a block `Mass`.
- Double-click the `Force` block to open the ESDL editor.
- Click on the **Input** button to create two inputs `x` and `v` (type `continuous`).
- Click on the **Output** button to create an output `F` (type `continuous`).
- Click on the **Parameter** button to create the constants `c` (spring rate), `d` (attenuation constant) and `l0` (length of the spring at rest).

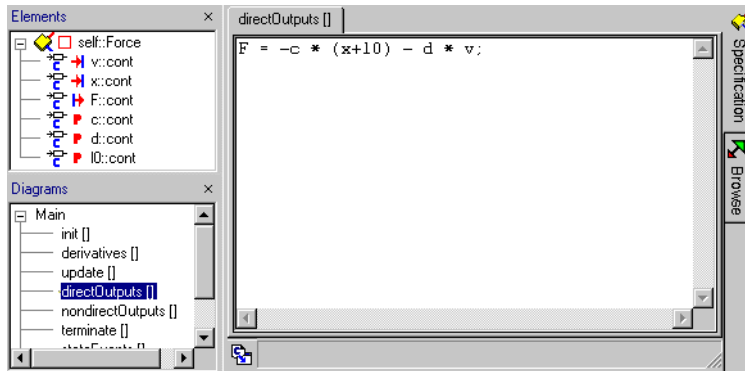


The methods in the "Diagram" pane are fixed by default.

- Click on each constant in the "Element" pane in turn to highlight it.
 - Right-click the highlighted constant to open the context menu.
 - Select the **Edit Data** menu item.
- The "Numeric Editor" dialog box opens.

- Assign realistic values to the constants (e.g., 5.0 to the spring rate c , 1.0 to the attenuation constant d , and 2.0 to the length of the spring at rest l_0).
- Click in the "Diagrams" pane on the method `directOutputs[]` and in the "Edit" field, specify the formula used to calculate the force:

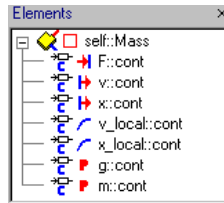
$$F = -c * (x + l_0) - d*v;$$



- Click on the **Generate Code** button.
The CT block `Force` is compiled.
- Double-click the `Mass` block to open the ESDL editor.
- As above, create an input F , two outputs x and v , one parameter m (mass), and one constant g (gravitational acceleration).
- Assign values to g and m as described above (9.81 to g and, e.g., 2.0 to the mass m).



- Click on the **Continuous State** button to create state variables `x_local` and `v_local` for the internal calculation of the outputs.

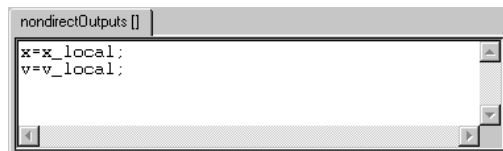


- For the `derivatives[]` method, specify the differential equations required for the calculation:

```
x_local.ddt(v_local)
v_local.ddt(-g + F/m)
```



- In `nondirectOutputs[]` pass the state variables `x_local` and `v_local` to the outputs `x` and `v`.



- In the `init[]` method, you can provide the system with realistic initial values for `x` and `v` using the `resetContinuousState()` function.

```
init []
resetContinuousState(x_local, 0.0);
resetContinuousState(v_local, 0.0);
```



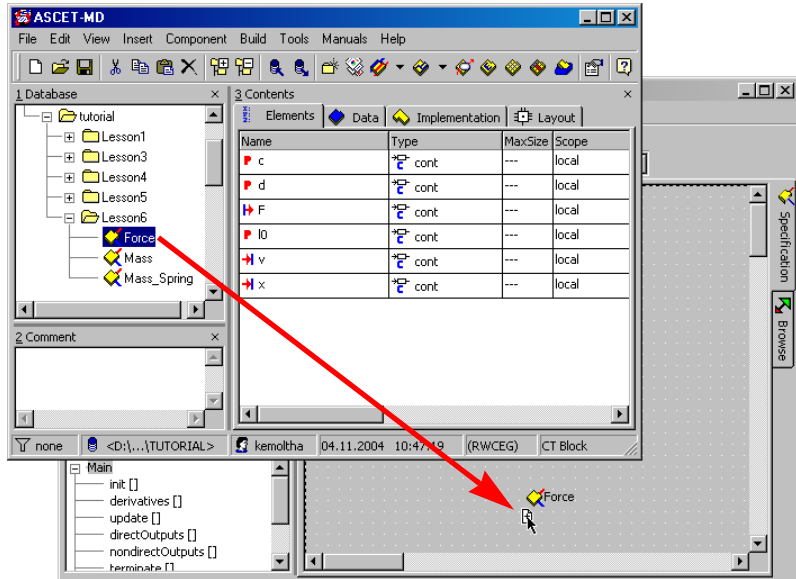
- Click on the **Generate Code** button.
The CT block `Mass` is compiled.

The combination of the two basic CT blocks into one CT structure block is done using the Block Diagram Editor (BDE).

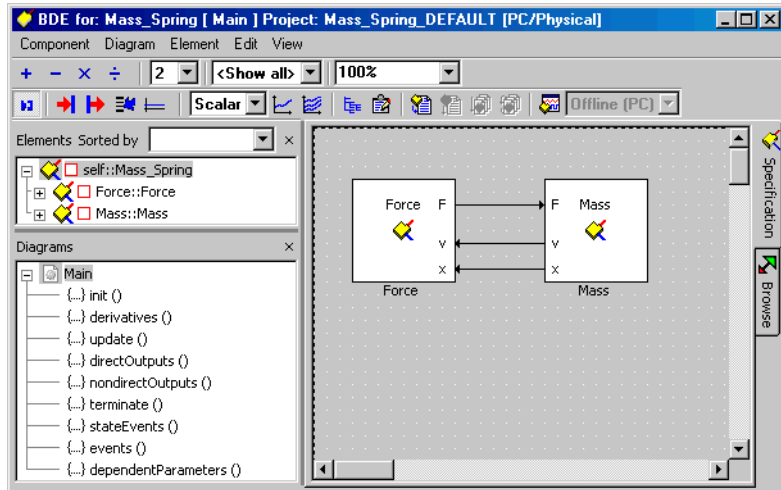
To combine the two basic CT blocks.

- For this purpose, go to your directory in the Component Manager.
- Select **Insert** → **Continuous Time Block** → **Block Diagram** to create a new block `Mass_Spring`.
- Double-click the new block to open it in the Block Diagram Editor.

- In the Component Manager, drag and drop the Mass and Force blocks (one at a time) to the BDE window and file them.



- Connect the corresponding inputs and outputs with each other.



Note

Double-clicking one of the CT basic blocks makes it available for editing. Note, however, that any modification to the blocks affects the entire library, i.e., all structure blocks that use these basic blocks.

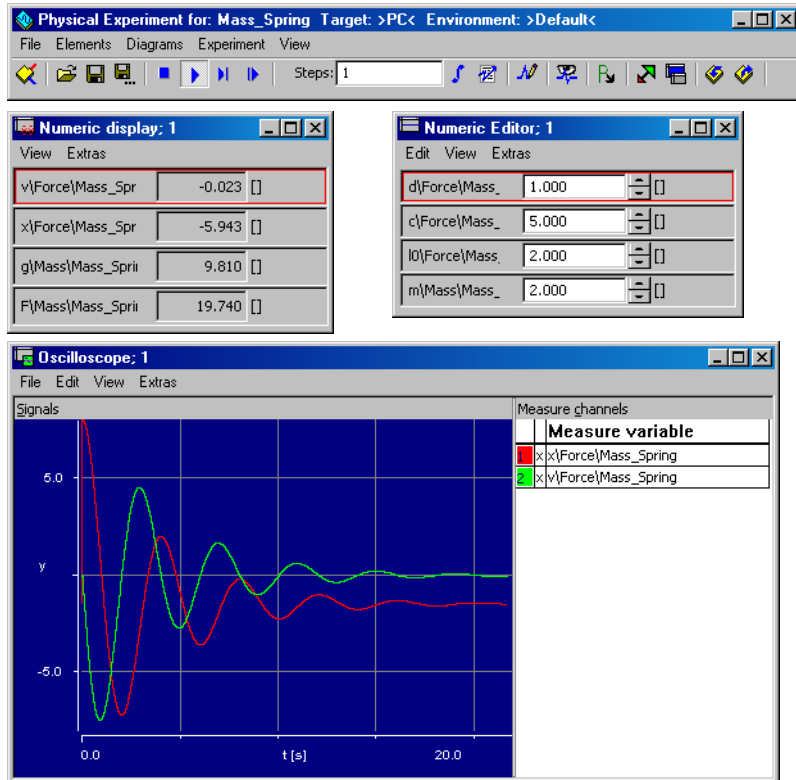


Open Experiment for selected Experiment Target

- Click on the **Open Experiment for selected Experiment Target** button.

The CT block is now compiled, and the experiment is started.

- Create the experimentation environment required with numeric editors for the parameters and graphical displays.



- Scale the channels in the oscilloscope separately, from -10 to 0 for x, from -8 to +8 for v.

6.6.3 Summary

After finishing this lesson, you should be able to carry out the following tasks in ASCET:

- Creating a model to simulate a process
- Using the ESDL editor to create CT blocks with direct and non-direct pass-through
- Using the Block Diagram Editor to combine CT blocks
- Performing the physical experiment

6.7 A Process Model

Following the introduction of CT blocks in the last chapter, you will now use them for testing your controller. In ASCET you can develop a model of the technical process to be controlled, and then experiment with a closed control loop. This means that way the controller can be thoroughly tested before it is used in a real vehicle.

In our example here, the motor is the technical process. It returns a value `u_n` which is a sensor reading of the rotational speed of the engine. This value is processed by the controller, which returns a value `air_nominal`. The controller output value determines the throttle- position of the engine, and thus in turn influences the rotational speed.

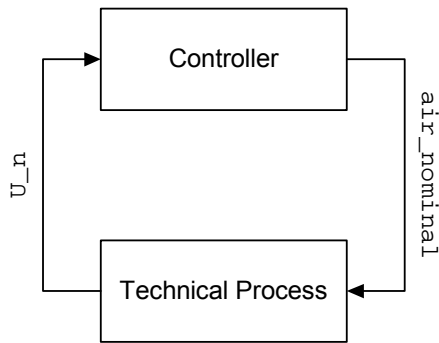


Fig. 6-1 A closed-loop experiment

You will use a CT block for this process model. This type of component is particularly suitable for process models. The model is based on the following differential equation, which models a PT2 - system:

$$T^2 s'' + 2DTs' + s = Ku$$

Equ. 6-1 A PT2 - system

The parameters T , D and K have to be set up with appropriate values.

6.7.1 Specifying the Process Model

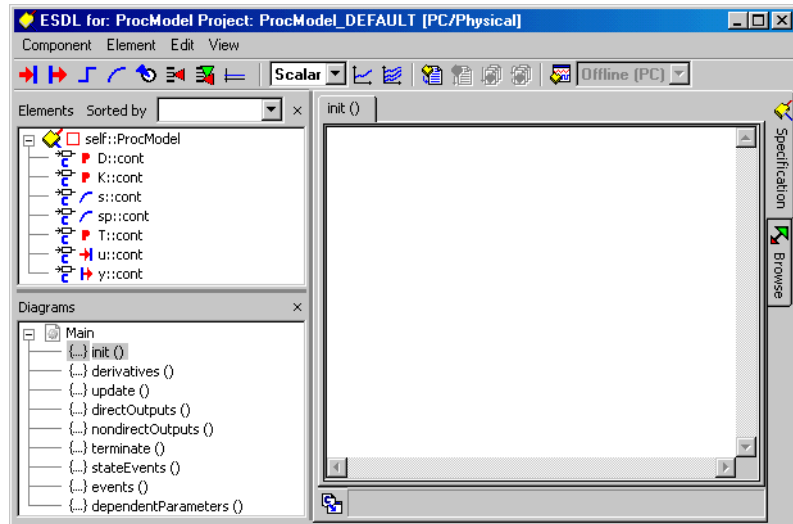
Creating continuous time components is different from creating other components. They have inputs and outputs which are the equivalent of arguments and return values. The main difference is that a continuous time block can have multiple inputs and outputs which are not tied to a particular method. There is a fixed set of methods defined in each continuous time block, that cannot be modified by the user.

You will use ESDL Code for the example here. The syntax of the ESDL code is similar to C++ or Java. An object method is called with the name of the object, a dot, the name of the method and the arguments in brackets followed by a semicolon. The method used for deriving is called `dat()`. For example, the equation $sp = \dot{s}$ is equivalent to the ESDL statement `s.dat(sp) ;`.

To create a continuous time component:

- In the Component Manager create the folder `Tutorial\Lesson7`.
- To add a continuous time block, select **Insert** → **Continuous Time Block** → **ESDL**.
- Name the new component `ProcModel`.
- Select **Component** → **Edit** to open the ESDL editor.

You can, of course, also use the external text editor. There are instructions for this in the first part of the tutorial.



To edit the process model, first add the elements required and then edit the methods `derivatives` and `non directOutput`.

To edit the process model:



- In the ESDL editor, use the **Continuous State** button to create two continuous states.

- Name the states s and sp .



- Create an input by clicking the **Input** button.

- Name the input u .



- Create an output by clicking the **Output** button.

- Name the output y .

Both elements are of type `cont`.

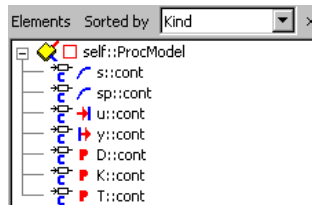


- Create a parameter by clicking on the **Parameter** button.

- Name the parameter D .

- Create the parameters κ and T .

The "Elements" list for the process model should look like this:



- Adjust the parameters as follows:

$$D = 0.4,$$

$$\kappa = 0.002,$$

$$T = 0.05.$$

- In the "Diagrams" list, select the `derivatives` method and edit the code as follows:

```
derivatives []
s.ddt(sp);
sp.ddt((K*u - 2*D*T*sp - s)/(T*T));
```

The illustration shows the internal text editor.

Note

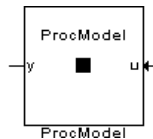
See Tab. 9-1 in chapter 9.1 of the ASCET reference guide for information on how to resolve a differential equation.

- In the "Elements" list, select the `nondirectOutputs` method and type in the following text.

```
nondirectOutputs []
y = s;
```

- Adjust the layout in the layout editor.

Note that in a process model it is preferable to put the outputs on the left and the inputs on the right.



- Select **Edit** → **Save**.
- In the Component Manager, click on the **Save** button to save the process model.



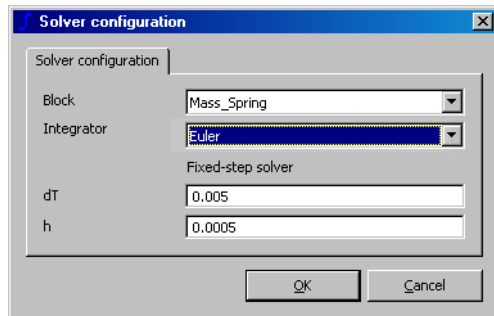
You can now start experimenting with the new model.

To experiment with the model:

- In the ESDL editor, select **Component** → **Open Experiment** to open the experimentation environment.



- Click on the **Open CT Solver** button to open the "Solver Configuration" dialog pane.
The configuration is displayed as follows:



- Click on **OK** to accept the default configuration.
- Open the data generator and create a channel for the input u .
- Set up the channel u with the following values:

Mode: pulse
Frequency: 0,5 Hz
Phase: 0.0 s
Offset: -0.5
Amplitude: 1.0

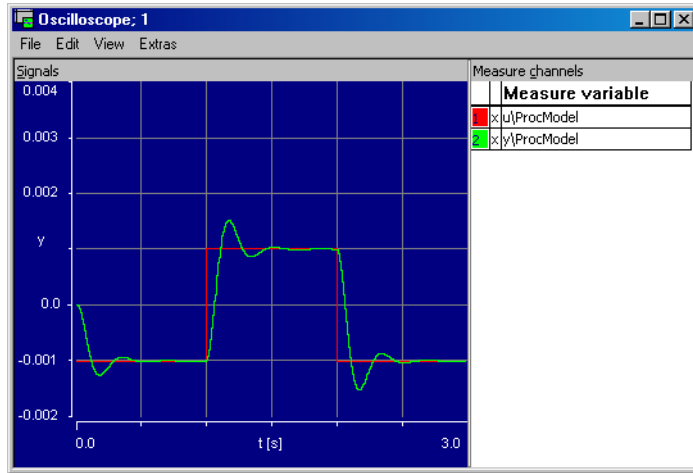
- Open an oscilloscope window with the channels u and y .
- Set the Measure Channels for the oscilloscope as follows:

	u	y
Min	-1	-0.002
Max	2	0.004
Extent	3.0	3.0



- Click on the **Save Environment** button.

- Start the experiment.
The output should look like this:



6.7.2 Integrating the Process Model

To create a closed control loop, we will now integrate the process model into the controller project we created earlier. The steps required are the same as before: including the module, setting up the operating system and linking the global elements.

Note

The process model is added to the same project for simplicity. This is often useful in the early stages of testing closed loop simulation. In regular projects, the process model would be distributed over a network in another project since they are not part of the same embedded system.

To include the process model:

- From the Component Manager, open the project editor for `ControllerTest`.
- In the project editor, add the component `ProcModel` to the "Elements" list.
- Activate the "OS" tab of the project editor to specify the scheduling for the CT tasks.



- Select the task `simulate_CT1` and set the value in the "Period" field to 0.01 s.
The controller and the process model both run in the same time interval.

Linking the continuous time blocks and the modules cannot be done automatically. They have to be connected explicitly in a block diagram.

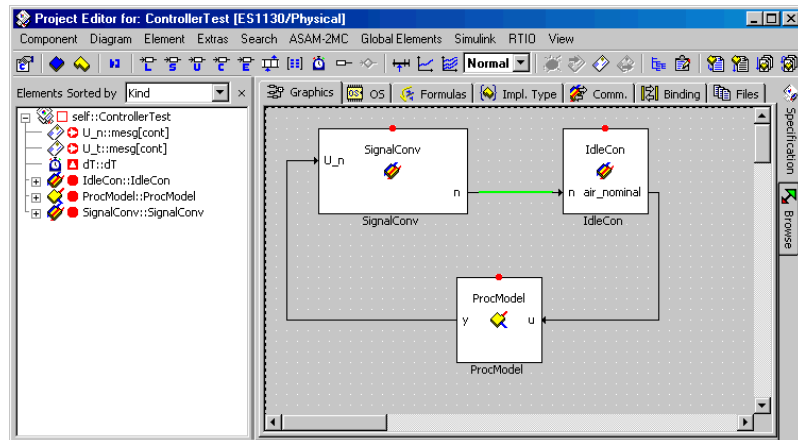
To adjust the linking between modules and CT block:



- Click the "Graphics" tab.
- From the "Elements" list, drag the three components and drop them into the drawing area.
- Connect the messages of the modules with the corresponding input and output of the CT block.

To construct the example, connect the output `y` of `ProcModel` with the global message `U_n` and connect the input `u` of `ProcModel` with the global message `air_nominal`.

- Right-click on each component and select **Unconnected Ports** to remove these ports from the diagram.



Linking the messages for communicating between modules is done automatically. Messages that have the same name are linked with each other.

The project is now complete and ready for experimentation. We will now experiment online, which requires an ASCET-RP installation and a real-time target (e.g. ES1000). If you do not have both, you will have to continue by experimenting offline as before.

Note

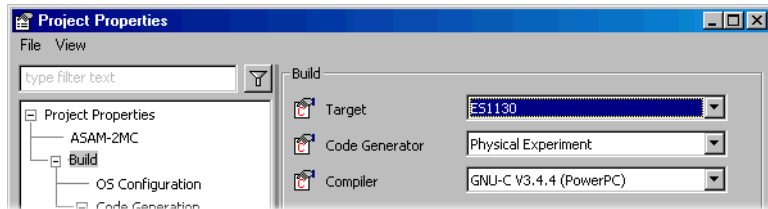
If you continue by experimenting offline, be sure to remove the global message `U_n` from the data generator.

To set up the project for online experimentation:



- Click on the **Specify Code Generation Options** button.
- In the "Settings" dialog window, "Build" tab, select the target **ES1130** and the **GNU-C (PowerPC)** compiler.

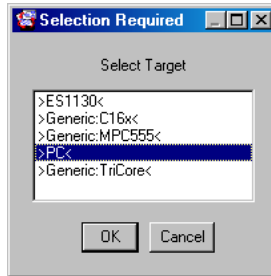
These options specify the hardware and the corresponding compiler for code generation.



- Click **OK** to close the dialog box.
The buttons **Open Experiment for selected Experiment Target** and **Reconnect to Experiment of selected Experiment Target** are now available.
- Click on the "OS" tab to activate the operating system editor.



- To copy the schedule you created earlier, select **Operating System** → **Copy From Target**.



- From the "Selection Required" dialog, select **PC** and click **OK**.

The project for the new target now has the same scheduling as that specified before for the offline PC simulation.

There are several differences from the offline experiment. In the online experiment, there is no event or data generator. The event generator serves to simulate the scheduling of the operating system tasks generated for online experiments.

In the online experiment the experimentation code and the measurements are started separately, and have separate buttons in the toolbar. This is because the measurements may influence the real-time behavior of the experiment, so it may sometimes be necessary to switch them off.

To experiment with the project online:



- Select **Online (RP)** from the "Experiment Target" combo box.

Offline (RP) is intended for offline experiments on the Target.

- Select **Component** → **Open Experiment**

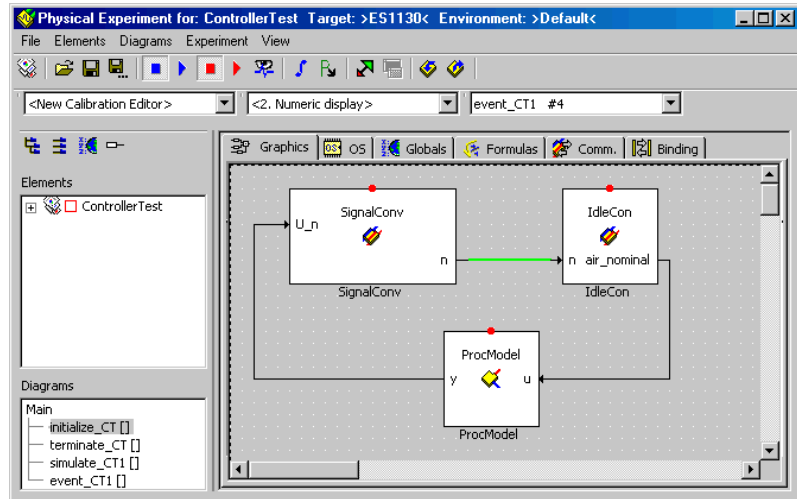
or



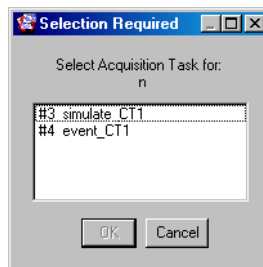
Open Experiment for selected Experiment Target

- click the **Open Experiment for selected Experiment Target** button.

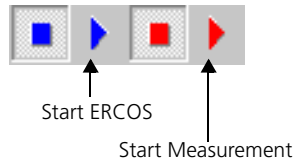
The code for the experiment is generated and the experiment opens with the same environment as defined previously.



If your project contains several tasks, you could well be prompted to select one acquisition task for each measure value.



- In the "Selection Required" window, select the #3 `simulate_CT1` task and click **OK**.
- Include `n` and `n_nominal` in the existing oscilloscope and set their value range from 0 to 2000.



- Open numeric editors for the variables $n_{nominal}$, κ_i and κ_p .
- Click on the **Start ERCOS** button and then click on the **Start Measurement** button.
The experiment starts and the results are displayed on the oscilloscope. The value for n should quickly approach $n_{nominal}$ and stay there.
- Modify $n_{nominal}$ in the numeric Editor.
The value n should change in line with all the changes to $n_{nominal}$.
- You can optimize the behavior of the control loop by adjusting the κ_i and κ_p parameters.

6.7.3 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

- Creating and specifying continuous time blocks
- Experimenting with continuous time blocks
- Integrating continuous time blocks in a project
- Creating variable links
- Switching between different targets
- Experimenting online with a project

6.8 State Machines

State machines are useful for modeling systems that move between a limited number of distinct states. ASCET provides a powerful mechanism for specifying components as state machines. In this lesson we will specify and test a simple state machine that implements a temperature dependent change in the nominal number of revolutions of an idling engine. That state machine will then be integrated into our project. In the next chapter we will then construct hierarchical state machines.

If the engine is cold, it has to idle at a higher speed to keep it turning over. Once the engine has warmed up, the rotational speed for idling can be decreased to reduce fuel consumption. Our state machine thus has two states: one when the engine is cold, and one when it is warm. It represents a two-phase control.

6.8.1 Specifying the State Machine

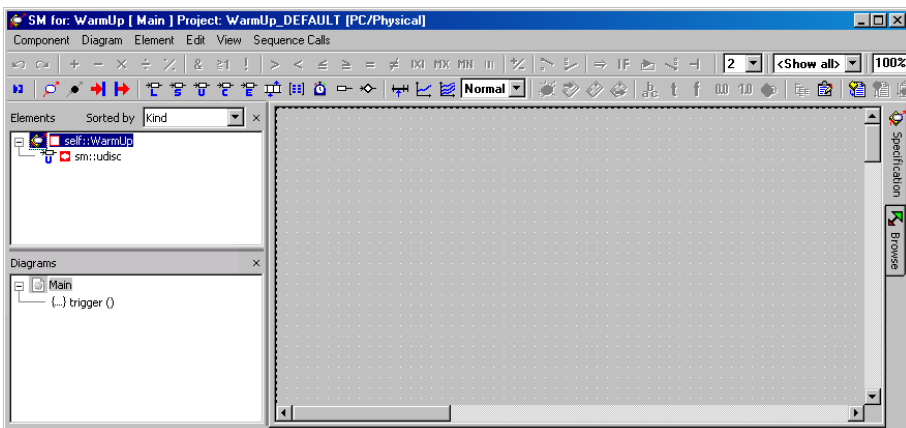
A state machine consists of the state graph itself and a number of specifications of actions and conditions. The actions and conditions can be specified using either block diagrams or ESDL code. They determine what happens in the various states and during the transitions between states.

The diagrams are specified in the block diagram editor. Another possibility is to write ESDL code directly in a text editor which can be opened for every state and every transition (i.e., *without* opening the ESDL editor). State machines have inputs and outputs for data transfer with other components.

To create a state machine:



- In the Component Manager, create the folder `Tutorial\Lesson8`.
- Select **Insert** → **State Machine** or click on the **Statemachine** button to create a new state machine.
- Name it `WarmUp`.
- In the "1 Database" list, double-click on the name of the state machine to open the state machine editor.



When you create a state machine, you specify the state graph diagram first and then define the various actions and conditions associated with states and state transitions.

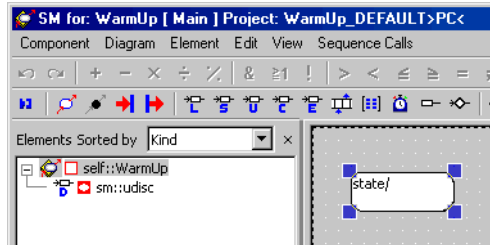
The state machine controlling your motor has two states: one for when the motor is cold and one for when the motor is warm.

To specify the state diagram:

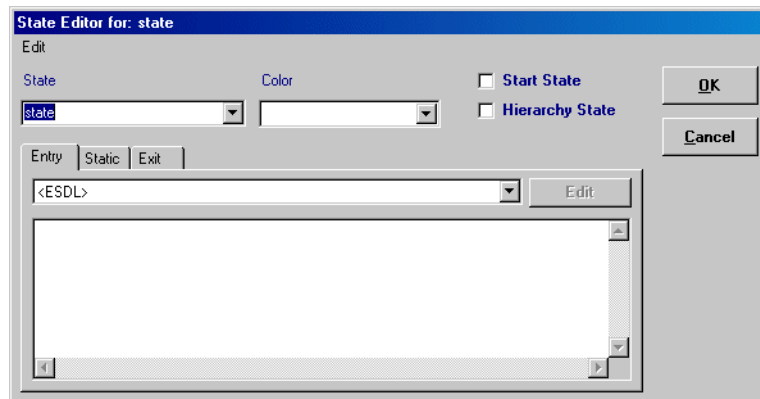


- Click on the **State** button to load the cursor with a state item.
- Click inside the drawing area, where you want to place the state.

A state symbol is drawn where you clicked.

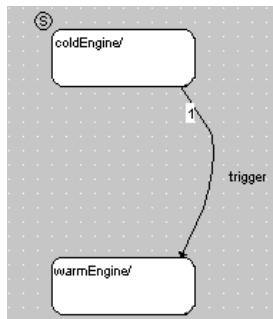


- Create a second state symbol and place it below the first one in the drawing area.
- Right-click on the state symbol you created first (the one on top) and choose **Edit State** from the context menu to open the State Editor.



- In the "State" field, enter the name `coldEngine`.
- Tick the **Start State** option to determine the state the machine is in when it is first started. There must be start state for each state machine.
- Click on **OK** to close the State Editor. The name is displayed in the state symbol.
- Name the second state symbol `warmEngine`.
- Right-click in the drawing area, outside any symbol, to activate the connection mode.
- Click in the right half of the `coldEngine` state symbol to begin a connection, then click in the right half of the `warmEngine` state symbol to connect the two states.

A line is drawn between the two state symbols. It has an arrow at one end, pointing from the top to the bottom symbol. The lines represent possible transitions between states.



- Create another transition from left half of the bottom to the left half of the top symbol.
- Select **Diagram** → **Store to Cache** to store the diagram.
- In the Component Manager, select **File** → **Save Database** to save the database.

The next step in building the state machine is to specify its interface. You need an input for the temperature value and an output for the number of revolutions. In addition, parameters are required that specify high and low temperature and number of revolutions per minute.

To specify the interface of the state machine:



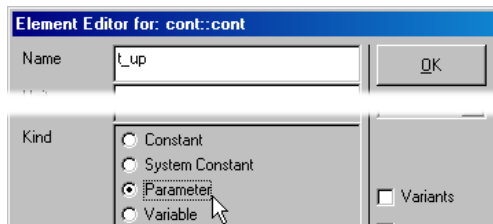
- Create an input by clicking the **Input** button.



- Create an output by clicking on the **Output** button.



- Name the output `n_nominal`.
- Click on the Continuous button to create a variable.
- In the element editor, enter the name `t_up` and activate the **Parameter** option.



With that, the originally created variable has become a parameter.

- Create three other parameters by the same method.
- Name the parameters and set their values as follows:

`t_up = 70`

`t_down = 60`

`n_cold = 900`

`n_warm = 600`

You can now proceed by specifying the actions and conditions for both the states and the transitions between states. You can specify three actions for each state:

- The entry action is executed each time the state is entered.

- The exit action is executed each time the state is left.
- The static action is executed while the state machine remains stationary.

Similarly, a trigger event, a condition, a priority and an action can be specified for each transition. The name of the trigger and of the condition appear next to the transition. *One* trigger is automatically created when the state machine is created.

The actions and conditions are specified in ordinary diagrams or in ESDL code. In this example you will use ESDL code.

To specify the trigger actions and conditions:

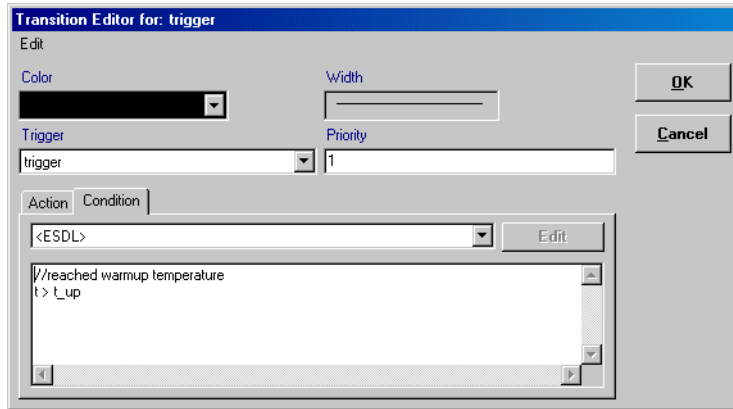
- Right-click on the transition from the `cold-Engine` state to `warmEngine`.
- From the context menu, select **Edit Transition** to open the Transition Editor.

The condition for a transition from cold to warm is that the actual temperature value t is greater than t_{up} .

- On the "Condition" tab, select `<ESDL>` from the combo box.

Note that you can influence the predefined choice of options in this combo box.

- Enter the code shown below in the code pane of the condition:



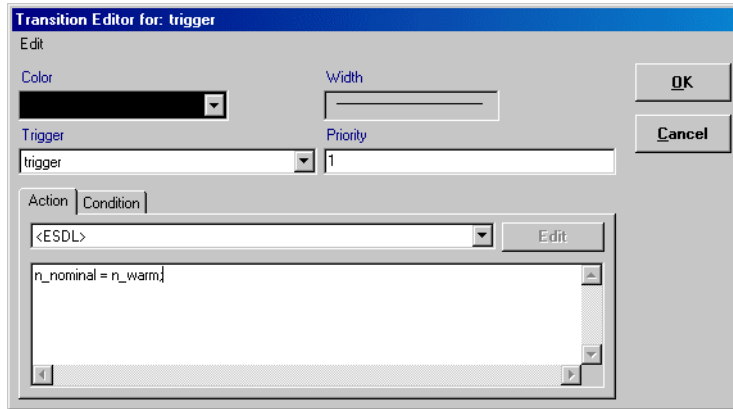
Note

In the Transition Editor, the condition is not terminated with a semicolon. This is also true for regular ESDL code where conditions appear in parentheses.

If the condition evaluates to `true`, the idle speed of the engine is set to `n_warm`.

Note that this code is displayed in the state machine diagram. In this example, an alias name is created for the transition condition and shown in the diagram.

- Select <ESDL> for the action, too, and enter the following code:



- Click **OK** to close the Transition Editor.
- Look at the diagram. Note that the condition and the action from the state machine can be seen.
- Open another editor for the transition from warmEngine to coldEngine.
- Select <ESDL> for the condition and enter the following code:

```
t < t_down
```

Note that this time the complete code is shown in the diagram as no alias was assigned (in a comment).

- Select <ESDL> for the action, too, and enter the following code:

```
n_nominal = n_cold;
```

- Close the editor and select **Diagram** → **Store to Cache**.

You can also specify the actions and conditions as block diagrams instead of ESDL code. For that purpose, you first create a separate diagram for actions and conditions.

To create a diagram for actions/conditions:

- In the state machine editor, select **Diagram** → **Add Diagram** → **Actions/Conditions BDE**.

A diagram named ActionCondition_BDE is created in the "Diagrams" pane.

- Accept the default name.

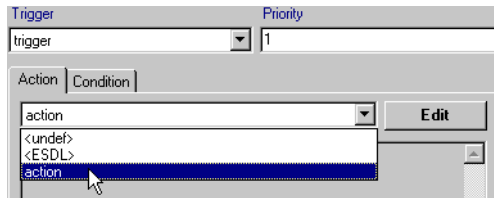


Now you can add, and specify, actions and conditions.

To specify the action/condition as block diagram:

- In the "Diagrams" field, click on the Action-Condition diagram.
- Use **Diagram** → **Add Action** or **Diagram** → **Add Condition** to create new actions and conditions.

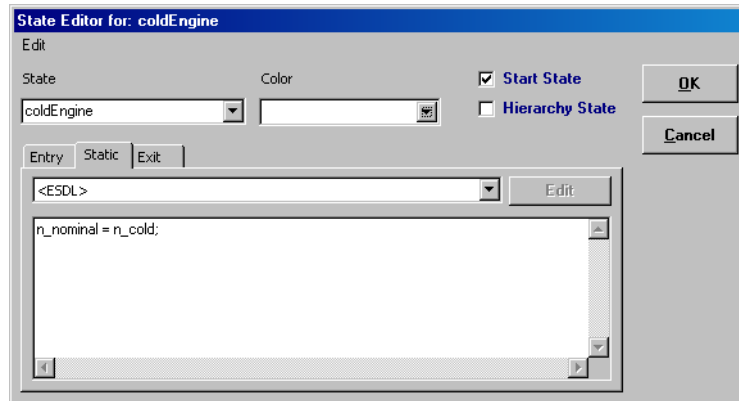
You can then select these actions and conditions from the combo boxes in the tabs in the "Transition Editor" dialog. Use the **Edit** button to go directly to the graphical specification in the BDE.



The initial value for the output `n_nominal` is still missing. Unlike the parameter values, this cannot be set. Instead you need to specify an action for the `coldEngine` start state. Since the entry action of the start state is *not* executed at the first activation of a state machine, you have to specify the initial value in the static action.

To specify an entry action:

- Right-click on the `coldEngine` start state.
- From the context menu, select **Edit State** to open the State Editor.



- Select `<ESDL>` from the combo box on the "Static" tab to specify the entry action.
Note that you can influence the predefined choice in this combo box via the "Defaults" tab in the "Options" window of the Component Manager.
- Enter `n_nominal = n_cold;` in the code pane to set the initial value of `n_nominal` to 900.
- Click on **OK** to close the state editor.

That completes the specification of your state machine. Before you start experimenting with it, you should understand the way it works.

6.8.2 How a State Machine Works

While it is usually easy to understand what a standard component does from its graphical specification, the function of a state machine may, at first, be less obvious. This section explains the principles of state machines using the example from the previous section. A detailed description of state machines and their functionality is given in chapter 2.5 "State Machines" of the ASCET reference guide.

Each state of a state machine has a name, an entry action, a static action and an exit action. It has transitions to and from other states. Each transition has a priority, a trigger, an action and a condition. All actions are optional.

Each state machine needs a start state. When the state machine is first called up, it is in the start state. It then checks the conditions in all the transitions pointing away from it. In our example there is just one such transition with the condition $t > t_{up}$. This condition checks whether the input value exceeds the value of the t_{up} parameter. If that is the case, the condition is true, and a transition takes place.

The parameters t_{up} and t_{down} determine the temperature that the engine has to reach, before the nominal rotational speed can be changed. In our example, if the engine temperature rises above 70 degrees, the speed can be reduced to 600 revolutions per minute. If it then falls below 60 degrees, the nominal speed must be reset to 900 revolutions per minute.

Whenever a transition takes place, the transition action specified for the transition is executed. In our example the transition action $n_{nominal} = n_{warm}$, which is executed when a transition from the state `coldEngine` to `warmEngine` takes place, sets the variable $n_{nominal}$ to 600. The transition action $n_{nominal} = n_{cold}$ sets it to 900 in the reverse case. When a transition occurs, the state machine also executes the exit action of the state it leaves, and the entry action of the state it enters. In our example, these are empty and nothing happens.

Once the state machine has entered the second state, it stays in that state until the condition in the transition from the second to the first state is fulfilled. While the state machine stays in one state, the static action is executed every time the state machine is triggered. Triggering is always an outside event which starts *one* pass through the state machine.

A pass through a state machine consists of first testing all the conditions on transitions leading away from the current state. Transitions and their conditions are tested in order of their priorities. If a condition is true, the corresponding transition is performed and the exit, transition and entry actions are executed. Once the first condition checks out true, any other transitions leading from the same state but having lower priorities are not tested. If no condition is true, the machine remains in the current state and performs the static action once for each pass.

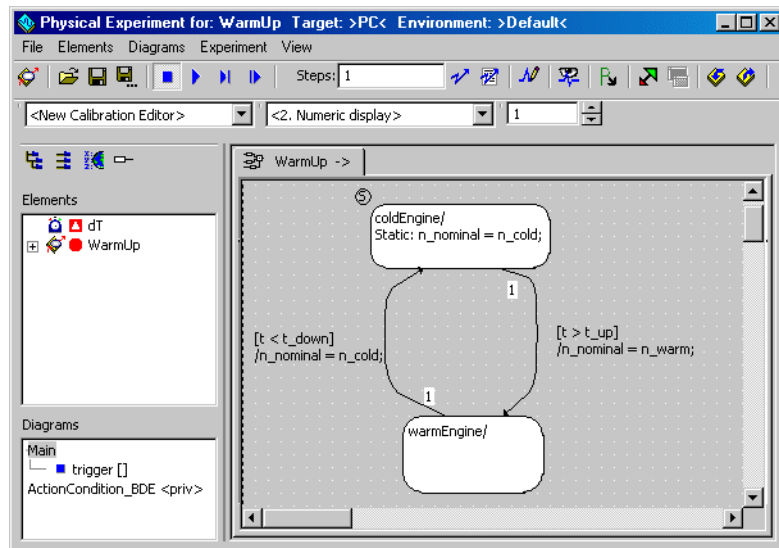
Once the condition in the second transition of our state machine is true, i.e. if the input value falls below the threshold, the state machine returns to the first state. The machine then remains in that state (doing nothing, because there is no static action) until the input value grows larger than the threshold again.

6.8.3 Experimenting with the State Machine

The experimentation environment works the same for state machines as for other types of components. One extra feature for experimenting with state machines is their animation, i.e. the current state is highlighted in the state machine diagram while the experiment is running.

To experiment with the state machine:

- In the state machine editor, select **Component** → **Offline Experiment** to open the experimentation environment.



- Right-click on one of the states and select **Animate States** from the context menu.
- Enable the `trigger` event.
- In the data generator, create a channel for the variable `t`.
- Assign a sine-wave with frequency 1 Hz, offset 70, and amplitude 20 to the channel.
- Open an oscilloscope window for `t` and `n_nominal`.



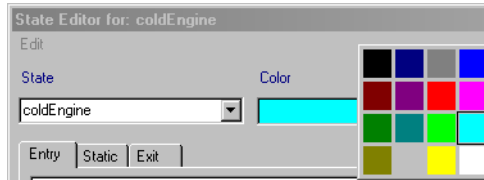
- Click on the **Start Offline Experiment** button to experiment with the state machine.

- Change the colors of the individual states to improve clarity.



- To do this, use the **Exit to Component** button to leave the experimentation environment, and call the state editor.

- Select the color in the "Color" combo box.



- Start the experiment anew.

The value of `n_nominal` changes according to whether the sine-wave exceeds or falls below the corresponding temperature threshold value. You can change the threshold using the calibration system to observe the effect of different values on the output. Also, in the state diagram the current state is highlighted.

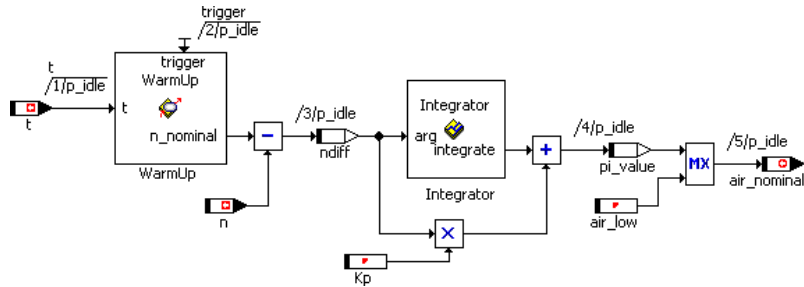
6.8.4 Integrating the State Machine in the Controller

Like all other components in ASCET, a state machine can be used as a building block within another component of any type. You can now integrate the state machine into your controller module to adjust the rotational speed to the engine temperature.

To integrate the state machine:

- From the Component Manager, open the module `Lesson4\IdleCon` in a block diagram editor.
- Remove the parameter `n_nominal` from the diagram and then from the "Elements" list. You will replace the parameter with the state machine in the block diagram.
- Select **Element** → **Add Item** and add the state machine to the "Elements" list of the controller.

- Create a receive message and name it t .
- Connect the output of the component `warmUp` with the subtraction operator in place of the deleted variable, and connect the input of `warmUp` with the receive message t .
- Adjust the diagram as shown below. Be sure to adjust the sequencing in the diagram to include all items in the correct order.



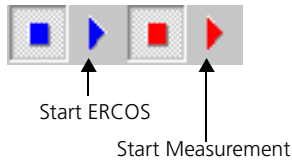
- Save the diagram and click on the **Save** button in the Component Manager.

In order to make the modified controller work with our project, we have to make some adjustments to the project. At this point we will also integrate the temperature sensor, which has been left unused so far.

To modify the project:

- Open the project editor for the project `ControllerTest`.
- Switch to the "OS" tab.
- Assign the process `t_sampling` to the task `Task10ms`.
- Use the command **Task** → **Move Up** to make the process `t_sampling` the first in that task.
- click the **Open Experiment for selected Experiment Target** button.
- Open an additional scalar calibration window for the value `U_t`.
- Add the variable t to the oscilloscope.





- Click on the **Start ERCOS** button.
- Click on the **Start Measurement** button.
- Adjust the value ϖ_t and observe its effect.

If the value of t exceeds the 70 degree limit, the state machine switches to nominal value for n to the lower value of 600. If the temperature falls to below 60 degrees (simulated by adjusting ϖ_t), the nominal value for n regains the original value of 900.

6.8.5 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

- Creating a state diagram
- Creating and assigning conditions, actions and triggers
- Experimenting with state machines
- Integrating state machines into other components

6.9 Hierarchical State Machines

Now that you have familiarized yourself with the way state machines work in the preceding lesson, we shall look at creating a more complex system. This unit concentrates on hierarchical state machines. You will also learn how to use the system libraries and components supplied with ASCET, such as timers.

ASCET permits structuring of state machines in closed and open hierarchies. With closed hierarchies, the internal functionality is concealed, with open hierarchies the substates are also shown graphically.

You will build a traffic light control system to run through the individual phases of a traffic light using parameterizable timing. The traffic light will also have an error status where it will flash.


6.9.1 Specifying the State Machine

First you will import the libraries you need and prepare for the task.

To import the system library `SystemLibETAS.exp`:



- In the Component Manager, click on **Import**. The "Select Import File" window opens.

- In the "Import File" field, use the  button to select the file `SystemLibETAS.exp` from the Export directory of your ASCET installation (e.g. `C:\etas\ASCET5.2\export`).

The **OK** button is now available.

- Click **OK** to start the import.

The "Import" window opens. All objects contained in the file are selected.

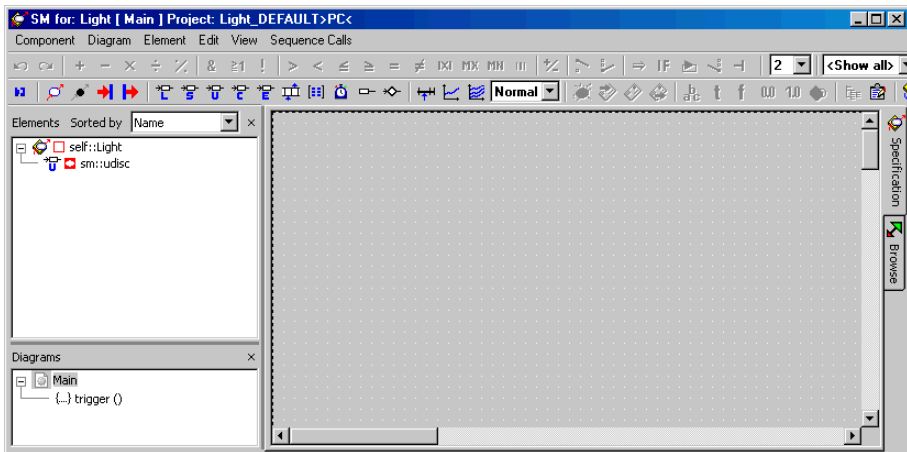
- Confirm the import of all files with **OK**.

The files are imported. This can take up to several minutes. When the import procedure is finished, all imported items are listed in the "Imported Items" window.

The second step is to specify the two main states possible for the traffic light (`NormalMode` and `ErrorMode`).

To create the state machine:

- In the Component Manager, create the folder `Tutorial\Lesson9`.
- Select **Insert** → **State Machine** to create a new state machine, and call it `Light`.
- Select **Component** → **Edit** to open the state machine editor.



You can start specifying the state machine that will control your traffic light.



- Create the two states `ErrorMode` and `NormalMode`.

Then add a timer to your project from the system library.

To add the timer object:

- Select **Element** → **Add Item**.
- In the "Select Item" dialog, select the timer object `Timer` from the `Counter_Timer` folder of the `SystemLib_ETAS` library.
- Confirm your selection with **Ok**.

You have now added an object `Timer` to the "Elements" list for your state machine.

To specify the state diagram:

- Specify the necessary data elements as follows:
 - An input error type `Logic`,
 - three outputs (`yellow`, `green`, `red`) type `Logic` to symbolize traffic light colors,
 - four continuous parameters (`BlinkTime`, `YellowTime`, `GreenTime`, `RedTime`) for the different traffic light phases.

To get more practice with dependent parameters, you will configure the parameters so that only the green phase is specified and the other parameters are given values dependent on that.

```
RedTime = 2 * GreenTime
YellowTime = GreenTime/3
BlinkTime = YellowTime/10
```

- Now specify calculations and dependencies of the individual parameters.

- To do this, select the check box **Dependent** under "Dependency" in the element editor for the parameters `RedTime`, `YellowTime` and `BlinkTime`.

The element editor is started with a double-click on the element name or via the **Edit** context menu.

- Click on the **Formula** button to start the formula editor.
- Using the formula editor, specify the calculation for each of the dependent parameters by first creating a formal parameter `x` and then entering the calculation in the formula pane.

```
Redtime : 2*x
YellowTime : x/3
BlinkTime : x/10
```

- Close the formula editor and the element editor.
- Open the dependency editor via the context menu **Edit Data**.
- Assign the corresponding model parameter to the formal parameter `x` for each of the dependent parameters.

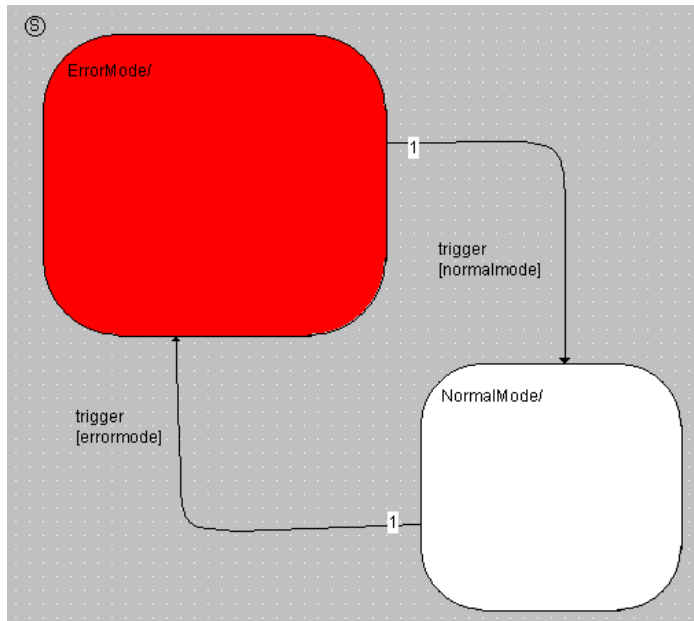
```
RedTime : x = GreenTime
YellowTime : x = GreenTime
BlinkTime : x = YellowTime
```

- Give the data elements meaningful values (e.g. `GreenTime = 5`).
- Open the state editor for the `ErrorMode` state.

You can open the state editor either by doubleclicking on a state or via the **Edit State** context menu.

- Define this state as the initial state and color it red.
- Enlarge both states so that the hierarchies can be inserted.
- Create the transitions between the two states.

- Open the transition editor via the **Edit Transition** context menu or with a double-click on the graphic.
- Specify the transitions between the two states by entering conditions in the transition dialog. Enter the conditions in ESDL so that the normal state `NormalMode` is activated when the input `error` is `false` (i.e. there has not been an error), and `ErrorMode` is activated when there is an error.



- Select **Diagram** → **Store to Cache**.
- Save your work in the Component Manager by selecting **File** → **Save Database**.
- You might like to experiment with the main states.

The next step towards creating the traffic light control system is to specify the substates. First specify the performance in the error mode (state `ErrorMode`). In this state, a yellow flashing light will be output. To do this, introduce two

substates `YellowOff` and `YellowOn`; with the timer as switch between them. In the `YellowOn` state, the output `yellow` will be set to `true`, while the `YellowOff` state sets it back to `false`.

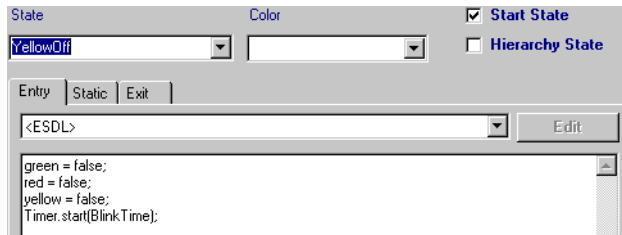
To specify the substates for the error mode



- Create the states `YellowOff` and `YellowOn` and place them inside the state `ErrorMode`.
- Define `YellowOff` as start state and color `YellowOn` yellow.
- Define the response of the state `YellowOff` in the state editor.

To do this, call the state editor either via the **Edit State** context menu or with a double-click on the state.

- For the entry action, select `ESDL` in the combo box for the "Entry" tab and enter the following code:



- For the static action, enter the following code on the "Static" tab:

```
Timer.compute();
```

- Now define and describe the `YellowOn` state in the same way.

Entry action:

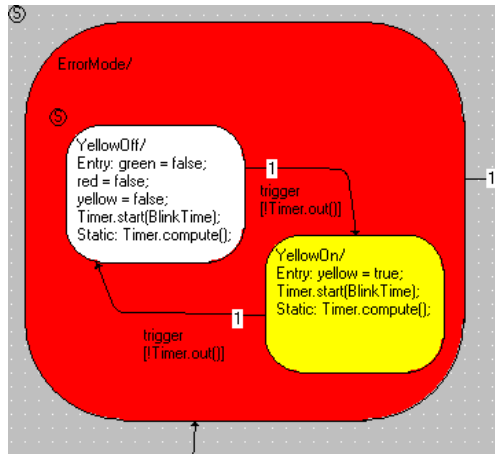
```
yellow = true;
Timer.start (BlinkTime);
```

Static action:

```
Timer.compute();
```

- Now define the transitions between the two substates.

The condition for a state transition is that the timer has run out (`Timer.out() == false`).



This means that the `ErrorMode` state is started in the `YellowOff` state. As well as switching off the color signals, the entry action starts the timer with the parameterizable flashing time. The static action of the `YellowOff` state calls the timer function `compute()` each time, which decrements the timer counter. When this counter is 0, the timer function `out()` returns the code `false`, thus fulfilling the transition condition. The state `YellowOn` works in a similar way, however, in the entry action, the `Yellow` color signal is switched on.

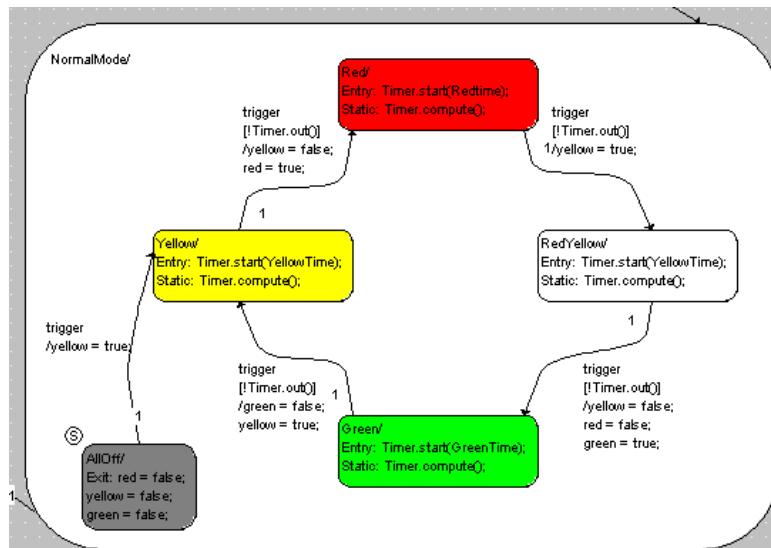
The next step is to specify the performance in normal operation. To do this, create a start state, `AllofF`, and place it within the `NormalMode` state. Use the exit action to set all the color signals to a defined state. Now think about a suitable response for the traffic light control system.

In this example, you should describe the activation or deactivation of the individual color signals in the transition actions, not in the entry actions of the states.

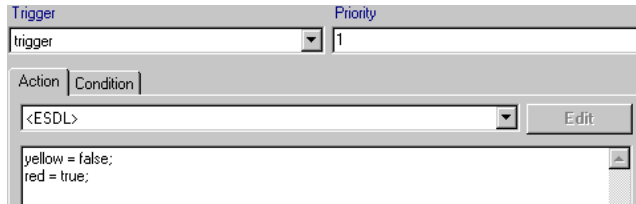
To specify the substates in normal operation

- Create and place the states `AllOff` (start state), `Yellow`, `Red`, `RedYellow` and `Green`.
- Specify the response for the states by starting the appropriate timer for each color (entry action) and initiating timer processing in the static action. (`Timer.compute()`).
- Define the state transitions and describe the response of the states within the transition actions.

The transition from `AllOff` to `Yellow` should generally occur, all other transitions should happen after the relevant timer has run out.



- Enter the actions for each color signal in the "Action" tab of the transition editor, e.g.



- Close the transition editor and select **Diagram → Store to Cache**.

That completes the specification of your traffic light control system. Before you can experiment with it, you should enter meaningful values for the parameters in the various color timers.

6.9.2 Experimenting with the Hierarchical State Machine

You can experiment with the hierarchical state machine in the same way as with the basic state machine. Please do not forget to activate the animation in the experiment.

Experimenting with the State Machine:

- In the state machine editor, select **Component → Open Experiment** to open the experimentation environment.
- Right-click on one of the states and select **Animate States** from the context menu.
- Enable the `trigger` event.



- Click on the **Start Offline Experiment** button to experiment with the state machine.

- Experiment with the state machine by changing the `GreenTime` parameter and then updating the dependent parameters via **Update Dependent Parameter**.



- Occasionally, set the `error` input to `true`.

6.9.3 How Hierarchical State Machines Work

Hierarchical state machines work in the same way as normal state machines. In principle, hierarchical state machines only represent a graphic structure of the total set of responses. As an extra task, consider or demonstrate how the response described could be achieved without a hierarchy.

The traffic light example is constructed with two hierarchical states. The system switches between the two states `ErrorMode` and `NormalMode` using the logical input variable `error`. The sub-responses are defined within these states.

To understand this, look at the processing in the `ErrorMode` hierarchy state. Each time the trigger is called, the condition for the transition from the hierarchy state `ErrorMode` to the hierarchy state `NormalMode` is checked (condition: `!error`). If no transition is necessary, the transitions from substate `YellowOff` to `YellowOn` or vice versa are checked, and the necessary actions are performed.

If you now look at `NormalMode`, this means that, again, for each trigger call it is first checked whether the input `error` is `true`, and therefore a transition to `ErrorMode` is necessary. Only if this is not the case, the transitions from the substates (`Alloff`, `Yellow`, `Red`, `RedYellow`, `Green`) are checked. In the traffic light example, it is checked whether the timer has run out.

You can have a look at the code generated from the state diagram to clarify this process.

Displaying generated code:

- In the state machine editor, select **Component** → **View Generated Code** to display the code generated.

The code from the components is written to a temporary file and then opened with an application defined in the operating system register database.

Note

In order to display the code generated, a search is made in the operating system register database for an application with associated files of type `.c` and `*.h`. Depending on the file endings registered, the relevant editor is started.*

6.9.4 Summary

After completing this lesson you should be able to perform the following tasks in ASCET:

- Create hierarchical state diagrams
- Describe the way the states behave in actions and also in the transition actions.
- Import modules, classes or components
- Import system components from ASCET libraries
- Use the Timer system component
- Use of dependent parameters
- Displaying generated code

7

Glossary

In this glossary the technical terms and abbreviations used in the ASCET documentation are explained. Many terms are also used in a more general sense, but only the meaning specific to ASCET is explained here.

The terms are listed in alphabetic order.

7.1

Abbreviations

ASAM-MCD

Association for **S**tandardisation of **A**utomation- and **M**easuring Systems, with the working groups **M**easuring, **C**alibration, **D**iagnosis (German: **A**rbeitskreis zur **S**tandardisierung von **A**utomations- und **M**esssystemen, mit den Arbeitsgruppen **M**essen, **C**alibrieren und **D**iagnose)

ASCET

Development tool for control unit software

ASCET-MD

ASCET Modeling and Design

ASCET-RP

ASCET Rapid Prototyping

ASCET-SE

ASCET Software Engineering

BDE

Block **D**iagram **E**ditor

CPU

Central **P**rocessing **U**nit

ECU

Embedded **C**ontrol **U**nit

ERCOS^{EK}

ETAS real-time operating system, OSEK-compliant

ESDL

Embedded **S**oftware **D**escription **L**anguage

ETK

emulator test probe (German: **E**mulator**t**ast**k**opf)

FPU

Floating **P**oint **U**nit

HTML

Hypertext **M**arkup **L**anguage

INCA

Integrated **C**alibration and **A**cquisition **S**ystems

INTECRIO

A new ETAS product family. INTECRIO integrates code from various behavioral modeling tools, facilitates all necessary configurations, allows the generation of executable code, and provides an experiment environment for the execution of the Rapid Prototyping experiment.

OS

Operating **S**ystem

OSEK

Working group "open systems for electronics in automobiles"
(German: Arbeitskreis **O**ffene **S**ysteme für die **E**lektronik im **K**raftfahrzeug)

RAM

Random **A**ccess **M**emory

ROM

Read-**O**nly **M**emory

UML

Unified **M**odeling **L**anguage

XML

Extensible **M**arkup **L**anguage

7.2

Terms

Action

An action is part of a state machine and associated with states or transitions of the state machine. An action is a piece of functionality, whose execution is triggered by the state machine.

Application Modes

An application mode is part of the operating system of ASCET. An operating mode describes different conditions a system can be in, e.g. EEPROM-programming mode, warm-up, or normal mode.

Argument

An argument is the input to a method of a class. Arguments can only be used in the specification of the method they belong to, and not in other methods of the class.

Arithmetic Services

User-defined C functions to optimize elementary operations, such as addition operations, and to extend such operations with special properties, such as value limits.

Array

An array is a one dimensional static list of elements of the basic scalar type `continuous` or `discrete`, indexed by the basic scalar type `discrete`.

ASAM-MCD-2MC file

Default exchange format used for projects in ASCII format for the description of measurement and calibration values. The files have the extension `*.a21`.

Basic Model Types

Basic model types are used to model physical behavior. There are three types: `continuous`, `discrete` and `logical`. A number of operations, such as addition or comparison, are defined for the basic model types. The implementation is used to transform the model types to implementation types.

Block Diagram

A block diagram is a graphical description for a component in which the various elements, operators and inputs/arguments and outputs/return values are connected by directed lines. A block diagram consists of several diagrams. The description in terms of block diagrams is a physical description in contrast to the description with C-Code.

Bypass Experiment

In a bypass experiment, ASCET is directly connected to a microcontroller, and parts of the microcontroller software are simulated by ASCET.

Calibration

Calibration is the manipulation of the values (physical / implementation) of elements during the execution of an ASCET model (experiment).

Calibration Window

ASCET working window which can be used to modify parameters.

C Code

C code is an implementation dependent description of a component.

Characteristic

General term used for characteristic map, curve and value (see also "Parameter").

Characteristic Line

Two-dimensional parameter.

Characteristic Map

Three-dimensional parameter.

Characteristic value

One-dimensional parameter (constant).

Class

A class is one of the component types in ASCET. Classes in ASCET are like object-oriented classes. The functionality of a class is described by methods.

Code

The executable code is the "actual" program with the exception of the data (contains the actual algorithms). The code is the program part which can be executed by the CPU.

Code Generation

Code generation is the first step in the transformation of a physical model to executable code. The physical model is transformed into ANSI C-Code. Since the C-Code is compiler (and therefore target) dependent, different code for each target is produced.

Component

A component is the basic unit of reusable functionality in ASCET. Components can be specified as classes, modules, or state machines. Each component is built up of elements which are combined with operators to build up the functionality.

Component Manager

Working environment in which the user can set up ASCET and manage the data he created and which are stored in the database.

Condition

A condition is used to describe the control flow in a state machine. It returns a logical value which determines, whether a transition from one state to another takes place.

Configuration dialog box

Dialog box used to configure the individual measuring and calibration windows as well as the variables contained therein.

Constant

A constant is an element that cannot be changed during execution of an ASCET model.

Container

Containers serve as containers for projects, classes and modules. Their purpose is to structure models and databases and place different database items under a common version control.

Data

The data is the variables of a program used for calibration.

Data Generator

The data generator is part of the experimentation environment. It is used to stimulate the inputs or variables in the model under experimentation.

Data Logger

With the data logger measurement data can be read from an experiment and stored to disk for further analysis.

Data Set

A data set contains/references the initial data for all elements of a component or project.

Database

All information specified or produced with ASCET is stored in a database. A database is structured into folders.

Description file

Contains the physical description of the characteristics and measured values in the control unit (names, addresses, conversion formulae, functional assignments, etc.).

Diagram

A diagram is used for the graphical specification of components as block diagrams or state machines.

Dimension

The dimension is used to describe the 'size' of basic elements. The dimension can either be scalar (zero dimensional), array (one dimensional) or characteristic line/table.

Distribution

A distribution contains the sample points for one or more group characteristic lines/maps.

Editor

See Calibration Window.

Element

An element is a part of a component which reads or writes data, for instance a variable, parameter or other component used within a component.

Event

An event is an (external) trigger that starts an action of the operating system, e.g., a task.

Event Generator

The event generator is part of the experimentation environment. It is used to describe the order and the timing in which events are generated for the activation of tasks (methods/processes/time frames) in the case of an offline experiment.

Experiment

An experiment defines the settings in the experiment environment that are used to test the proper functioning of components or projects. It contains information about the size, position and content of the measurement and calibration windows, as well as the settings of the event generator, data generator and the data logger. An experiment can be executed either offline (non real-time) or online (real-time) and can be used to control a technical process in a bypass or fullpass application. In all cases, instrumented code generated from an ASCET specification is used for experiment execution.

Experiment environment

Main working environment in which the user performs his experiments.

Fixed Point Code

From the physical specification, fixed point code can be generated which can be executed on processors without a floating point unit.

Folder

A folder is a management unit for structuring an ASCET database. A folder contains items of any kind.

Formula

A formula is part of an implementation describing the transformation from the model types to the implementation (data) types.

Fullpass Experiment

In a fullpass experiment, ASCET is directly connected with an experimental microcontroller, and the entire application is simulated by ASCET.

Group Characteristic Line/Map

Group characteristic lines/maps are characteristic lines/maps that share the same distribution of axis points but have different return values. The distribution of axis points and the individual group tables are specified as separate elements.

HEX file

Exchange format of a program version as Intel Hex or Motorola S Record file.

Hierarchy

A hierarchy block is used to structure the graphical specification of a block diagram.

Icon

Icons can be used to illustrate the function of ASCET components.

Implementation

An implementation describes the transformation of the physical specification (model) to executable fixed point code. An implementation consists of a (linear) transformation formula and a bounding interval for the model values.

Implementation Cast

Element that provides the users the possibility to control the implementations of intermediate results in arithmetic chains without changing the physical representation of the elements in question.

Implementation Data Types

Implementation data types are the data types of the underlying C programming language, e.g., `unsigned byte (uint8)`, `signed word (sint16)`, `float`.

Implementation Types

Implementation templates. Implementation types contain the main specifications of an implementation; they are defined in the project editor and can be assigned to individual elements in the implementation editors.

Intel Hex

Exchange format used for program versions.

Interface

An interface of a component describes how the component exchanges data with other components. It can be compared to the `.h` file in C.

Kind

There are three kinds of elements: variables, parameters, and constants. Variables can be read and written. Parameters can only be read but can be calibrated during experimentation. Constants can only be read and not written to during experiments.

L1

The message format for exchanging data between the host and the target, where the experiment is run. Data is transferred, e.g. for displaying values in measure windows.

Layout

A component has a graphical representation that shows pins for the inputs/arguments, outputs/return values and time frames/methods/processes. Additionally, the layout contains an icon that graphically represents the component when used within other components.

Literal

A literal is used in the description of components. A literal contains a string that is interpreted as a value, e.g. as a continuous or logical value.

Measuring

Recording of data which is either displayed or stored, or both displayed and stored.

Measure window

ASCET working window which displays measured signals during a measurement.

Measured signal

A variable to be measured.

Measurement

A measurement is the representation of values (physical / implementation) of variables/parameter during an experiment. The values can be displayed with various different measurement windows like oscilloscopes, numeric displays, etc.

Measuring channel parameters

Parameters which can be set for the individual channels of a measuring module.

Message

A message is a real time language construct of ASCET for protected data exchange between concurrent processes.

Method

A method is part of the description of the functionality of a class in terms of object oriented programming. A method has arguments and one return value.

Model Type

Each element of an ASCET component specification is either a component of its own or is of a model type. In contrast to implementation types, model types represent physical values.

Module

A module is one of the component types in ASCET. It describes a number of processes that can be activated by the operating system. A module cannot be used as a subcomponent within other components.

Monitor

With a monitor the data value of an element can be displayed in a diagram during an experiment.

Motorola-S-Record

Exchange format used for program versions.

Offline experiment

During offline experimentation the code generated by ASCET can be run on the PC or an experimental target, but it does not run in real-time. Offline experimentation focuses on testing the functional specification of a system.

Online experiment

In the online experiment the projects are executed in real-time with the behavior defined in the real-time operating system. The code always runs on an experimental target in real-time. The online experiment focuses on the operating system schedule and the corresponding real-time behavior of the control system.

Operating System

The operating system is used to schedule the execution/activation of an ASCET software system. The operating system also provides services for communication (messages) and access to reserved parts of the hardware (resources). The ASCET operating system is based on the real-time operating system ERCOS^{EK}.

Oscilloscope

An oscilloscope is a type of measurement window that graphically displays data values during experiments.

Parameter

A parameter (characteristic value, curve and map) is an element whose value cannot be changed by the calculations executed in an ASCET model. It can, however, be calibrated during an experiment.

Priority

Every task has a priority in the form of a number. The higher the number, the higher the priority. The priority determines the order in which tasks are scheduled.

Process

A process is a concurrently executable piece of functionality that is activated by the operating system. Processes are specified in modules and do not have any arguments/inputs or return values/outputs.

Program

A program consists of code and data and is executed as a unit by the CPU of the control unit.

Project

A project describes an entire embedded software system. It contains components which define the functionality, an operating system specification, and a binding mechanism which defines the communication.

Resource

A resource is used to model parts of an embedded system that can be used only mutually exclusively, e.g. timers. When such a part is accessed, it has to be reserved and then released again, which is done using resources.

Scheduling

Scheduling is the assigning of processes to tasks and the definition of task activation by the operating system.

Scope

An element has one of two scopes: local (only visible inside a component) or global (defined inside a project).

State

A state is a part of a state machine. A state machine is always in a one of its states. One of the states is marked as the start state which is the initial state of the state machine. Each state is connected to other states by arcs. A state has an entry action (that is executed upon entry of a state), an static action (that is executed the state remains unchanged) and an exit action (that is executed upon exit of the state).

State Machine

A state machine is one of the component types in ASCET. The behavior is described with a state graph consisting of states connected by transitions.

Target

A target is the hardware an experiment runs on. A target can either be an experimental target (PC, Transputer, PowerPC) or a microcontroller target.

Task

A task is an ordered collection of processes that can be activated by the operating system. Attributes of a task are its operating modes, its activation trigger, its priority, the mode of scheduling. On activation the processes of the task are executed in the given order.

Trigger

A trigger activates the execution of a task (in the scope of the operating system) or of a state machine.

Transition

A transition is a connection between states. Transitions describe possible state changes. Each transition is assigned to a trigger of the state machine, has a priority, a condition, and an action.

Type

Variables and parameters are of type `cont` (continuous), `udisc` (unsigned discrete), `sdisc` (signed discrete) or `log` (logic). `cont` is used for physical quantities that can assume any value; `udisc` for positive integer values, `sdisc` for negative integer values, and `log` is used for Boolean values (true or false).

User profile

A set of user-specific option settings.

Variable

A variable is an element that can be read and written during the execution of an ASCET model. The value of a variable can also be changed with the calibration system.

Also: General term used for parameters (characteristics) and measured signals.

Window elements

General term used for calibration and display elements.

8 Reference Lists

The chapter "Reference Lists" contains information on troubleshooting, the directory structure, and the reference files required. This chapter also includes a list of all keyboard commands sorted by working windows.

8.1 Troubleshooting and User Feedback

While developing ASCET, the functional safety of the program was utmost importance. Should an error occur nevertheless, please forward the following information to ETAS:

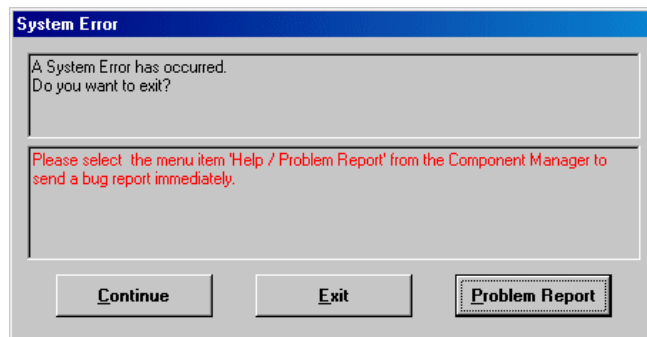
- Which step were you about to perform with ASCET when the error occurred?
- What kind of error occurred (wrong function, system error or system crash)?
- Which model element or model was edited at the time of the error?

Note

To allow ASCET to be updated and developed further, it is important that you report any errors which have occurred with an application to ETAS. You can use the "Problem Report" method for this purpose.

When you use the support function, ASCET compresses the entire contents of the "log" directory (all *.log files) including a textual description into an archive file named `EtasLogFiles00.zip` in the `...\ETAS\LogFiles\` subdirectory. For additional archive files, the file name is incremented automatically (up to 19) to avoid that older archive files are immediately overwritten.

If a critical system error occurs, the following window is displayed:



What to do in case of an error:

1. **Problem Report** button
 - Click on the **Problem Report** button.
The support function is started.
 - Describe the error and forward the information—together with the model—to ETAS.
2. **Exit** button
 - Click on the **Exit** button.
ASCET is closed; all modifications that have not been saved will be lost.
Close any message boxes prompting you to save data without saving any data.
 - Restart ASCET.
3. **Continue** button

Note

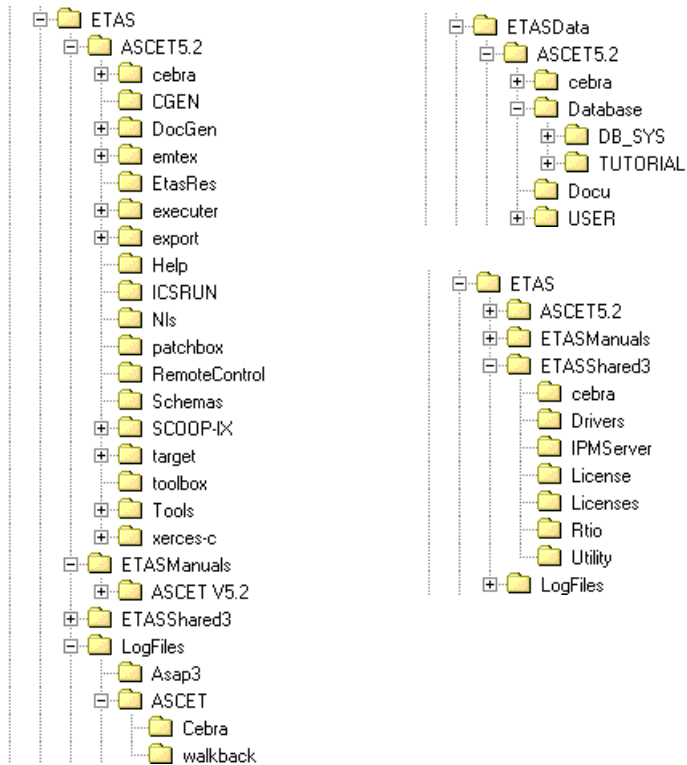
*Use the **Continue** button only if you have to save important configuration data. Subsequent errors or incorrect configurations cannot be excluded!*

- Click on the **Continue** button.
The application continues to run; the program jumps back to the location where it was before the error occurred.
- Save your data.
- Exit ASCET.
- Restart ASCET.

It is generally advisable to close the program (without saving) and to restart it. Thus, the risk of possible subsequent errors is omitted.

8.2 ASCET Directories

When installing ASCET, the following directory structure is created on the installation disk (unless you modify the path settings):



8.2.1 Default Storage Directories

- Database
ETASData\Ascet5.2\Database
- Export
ETAS\Ascet5.2\Export
- automatically generated documentation
ETASData\Ascet5.2\Docu

8.2.2 Changing Default Directories

You can use the "Options" dialog window to change the default storage directories. To do so, proceed as follows:

To change the default storage directories:

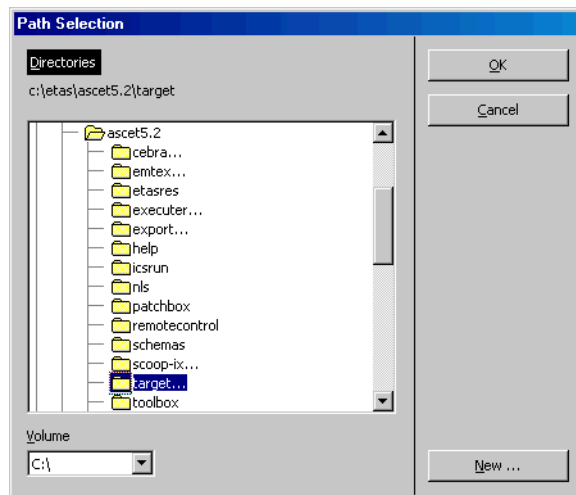
- In the Component Manager, select the menu option **Tools** → **Options**.

The ASCET options window opens. You can change the database path in the "Options" node, the export path in the "Export" node, and the documentation path in the "Documentation" node.



- Click on the button next to the path you want to change.

The "Path selection" window opens.



- Specify the directory you want to use as default for the selected option.

- Click **OK**.

The selected directory is displayed in the "Options" window.

- Repeat these steps for each option you want to change.

- When you are ready, click **OK** in the "Options" dialog to accept the changes, or click **Cancel** to discard the changes.

8.3 Keyboard Control

8.3.1 General Control Functions

Key	Function
ALT + TAB	Switches between open calibration programs.
ALT + SPACE	Opens the system menu of the calibration program window.
ALT + F4	Closes the current window.
CURSOR KEYS (← ↑ → ↓)	Moves to a table item or list item, ↓ also opens the active list field.
TAB	Moves the highlight (focus) to the next element (option) of a window (use SHIFT + TAB for the opposite direction).
DEL	Deletes a selected entry.
SPACE	Activates the input mode in tables, selects or deselects a table or list item.
ESC	Closes the input mode without accepting the entry.
SHIFT	Activates multiple selection; you can select a table area with the cursor keys while keeping the Shift key depressed.
ENTER	Confirms the entry and closes the input mode, and opens or closes branches.
CTRL + A	Selects all objects (e.g. in a list).
CTRL + C	copies data to the clipboard (except measure windows)
CTRL + V	inserts data from the clipboard
CTRL + X	cuts data to the clipboard
CTRL + Y	Repeats the last action. (for calibration windows, use CTRL + D instead).
CTRL + Z	Undoes the last action. (for calibration windows, use CTRL + U).
CTRL + F1	Shows the most important keyboard commands
F10	activate main menu

Tab. 8-1 General keyboard commands (in individual cases, a command can have a different meaning in a particular window)

8.3.2 Keyboard Commands in the Component Manager

The following special keyboard commands are available in the Component Manager:

Key	Function
CTRL + N	create a new database
CTRL + O	open database
CTRL + S	save database
CTRL + E	Activates the export function.
CTRL + M	Activates the import function.
ALT + F4	close Component Manager and exit ASCET
F2	rename selected object
CTRL + F	search a string in C code or ESDL components
CTRL + H	replace a string in C code or ESDL components
CTRL + Q	search the database from various points of view
F5	update Component Manager display
INSERT	insert folder / insert object in container
RETURN	open editor for selected database object
ALT + 1	switch to the "1 Database" field
ALT + 2	switch to the "2 Comment" field
ALT + 3	switch to the "3 Contents" field
ALT + F6	switch to the next window
ESC	cancel cut (<CTRL> + <X>)

8.3.3 Keyboard Commands in the Monitor Window

The following special keyboard commands are available in the ASCET monitor window:

Key	Function
Ctrl + o	open log file for the "Monitor" tab
Ctrl + s	save the content of the "Monitor" tab to a file
CTRL + F	find/replace text in the "Monitor" tab

Key	Function
CTRL + R	delete text in the "Monitor" tab
CTRL + +	enlarge monitor window
CTRL + -	scale down monitor window

8.3.4 Keyboard Commands in the Editors

The following special keyboard commands are available in the block diagram editor:

Key	Function
F2	rename selected element
CTRL + CURSOR RIGHT	show next sequence call
CTRL + CURSOR LEFT	show previous sequence call

The following special keyboard commands are available in the C Code and ESDL editors:

Key	Function
CTRL + F	find/replace
CTRL + S	save method/process

The following special keyboard commands are available in the AS editor:

Key	Function
CTRL + N	create new file
CTRL + O	open file
CTRL + S	save file

The following special keyboard commands are available in the data and implementation editors for components/projects:

Key	Function
ALT + C, ALT + O	close editor window
F2	rename selected data set/implementation

8.3.5 Keyboard Commands in the Offline Experiment Environment

The following special keyboard commands are available in the offline experiment environment:

Key	Function
ALT + F4	close offline experiment environment
F10	Activates the main menu
CTRL + C	calibrate element
CTRL + M	measure element
CTRL + S	stimulate element
CTRL + L	activates recording of selected elements in the Data Logger
CTRL + A	activates recording of all elements in the Data Logger
CTRL + I	view the implementation of an element
CTRL + U	update dependent parameters

8.3.6 Measure and Calibration Windows in General

The keyboard commands listed below equally apply to all measure and calibration windows. For specific keyboard commands for individual measure and calibration windows, please see further below.

Key	Function
CTRL + H	Displays hexadecimal values in the active window.
CTRL + I	Displays information on the selected variable.
CTRL + P	Displays physical values in the active window.
CTRL + S	Opens the display setup for the active window (except 3D graphical editor).
DEL	Deletes a variable from the active window (except graphical and numerical table editors).

Tab. 8-2 Keyboard commands in all measure and calibration windows

Calibration Windows

The following keyboard commands are—in addition to those in Tab. 8-2—available in all calibration windows:

Key	Function
CTRL + M	increments selected values (except 3D graphical editor)
CTRL + N	decrements selected values (except 3D graphical editor)
CTRL + D	Repeats the last action.
CTRL + U	Undoes the last action.

The following keyboard commands are only available in the *numerical editor*:

Key	Function
CTRL + R	displays binary values
CTRL + Z	displays decimal values
CTRL + PAGE UP	moves the highlighted variable in a window one position down
CTRL + PAGE DOWN	moves the highlighted variable in a window one position up

The following keyboard commands are only available in the *table editor*:

Key	Function
+	adds offset to selected values
*	multiplies selected values by a factor
=	fills selected cells with a value
CTRL + J	decrements the x-axis value (only characteristic line/map)
CTRL + K	increments the x-axis value (only characteristic line/map)
CTRL + R	decrements the y-axis value (only characteristic map)
CTRL + T	increments the y-axis value (only characteristic map)
CTRL + X	assigns a specific value to the x-axis point (only characteristic line/map)
CTRL + Y	assigns a specific value to the y-axis point (only characteristic map)

The following keyboard commands are only available in the *1D* or *2D graphical editor*:

Key	Function
X	Switches to the xz representation (2D Map Editor only).
Y	Switches to the yz representation (2D Map Editor only).
Z	Reverses x (y)-axis and z-axis.
CTRL + B	Allows several values on the curve to be selected.

The following keyboard commands are only available in the *3D graphical editor*:

Key	Function
CURSOR LEFT, CURSOR RIGHT	rotation around z-axis
CURSOR UP, CURSOR DOWN	rotation around horizontal axis
<NUM 4>, <NUM 6>	rotation around z-axis
<NUM 8>, <NUM 2>	rotation around horizontal axis

Measure Windows

The following keyboard commands are—in addition to those in Tab. 8-2—available in all measure windows:

Key	Function
CTRL + C	copies the settings of the current measure window to the clipboard
CTRL + W	copies the settings from the clipboard to the current measure window

The following keyboard commands are available in the *numerical display*, *bit display*, and *horizontal and vertical bar display*:

CTRL + PAGE UP	moves the highlighted variable in a window one position down (vertical bar display: to the left)
CTRL + PAGE DOWN	moves the highlighted variable in a window one position up (vertical bar display: to the right)

The following keyboard commands are only available in the *numerical display*:

Key	Function
CTRL + Z	Displays decimal values in the active window.
CTRL + R	Displays binary values in the active window.

The following keyboard commands are only available in the *oscilloscope and recorder window*:

Key	Function
CTRL + A	Adapts the Y-axis scaling for the selected measuring channel.
CTRL + U	Undoes the last scaling.
CTRL + L	Shows/hides the measuring channel list.
CTRL + X	Shows/hides the selected variable.
CTRL + V	Activates/deactivates the analysis mode.
CTRL + G	Shows/hides display grid (oscilloscope only).
T	Releases the trigger event manually.
PAGE DOWN	Selects the last channel in the "Measure channels" or "Bit channels" list.
PAGE UP	Selects the first channel in the "Measure channels" or "Bit channels" list.
CURSOR LEFT, CURSOR RIGHT	Move selected measure cursor in single steps (analysis mode only).
CTRL + CURSOR LEFT, CTRL + CURSOR RIGHT	Move selected measure cursor several steps at once (analysis mode only).

Windows XP comes with a built-in personal firewall. On many other systems it is very common to have personal firewall software from third party vendors, such as Symantec, McAfee or BlackIce installed.

Personal firewalls may interfere with access to Ethernet hardware using ASCET-RP or ASCET-SE. The automatic search for hardware typically cannot find any Ethernet hardware at all, although the configuration parameters are correct. In that case, you may have firewall software installed on your system.

This chapter helps you to configure the Windows XP firewall if the hardware access is prohibited under Windows XP with Service Pack 2.

The following actions in ETAS products may lead to some trouble if the Windows XP firewall is not properly parameterized:

- ASCET
 - opening an experiment
 - reconnecting to an experiment
- Hardware Service Pack
 - searching for hardware
 - starting a firmware update
- INCA
 - searching for hardware
 - opening the hardware configuration editor
 - opening an experiment

9.1 Users with Administrator Privileges

If you have administrator privileges on your PC, the following dialog window opens if the firewall blocks an ETAS product.



To unblock a product:

- In the "Windows Security Alert" dialog window, click on **Unblock**.

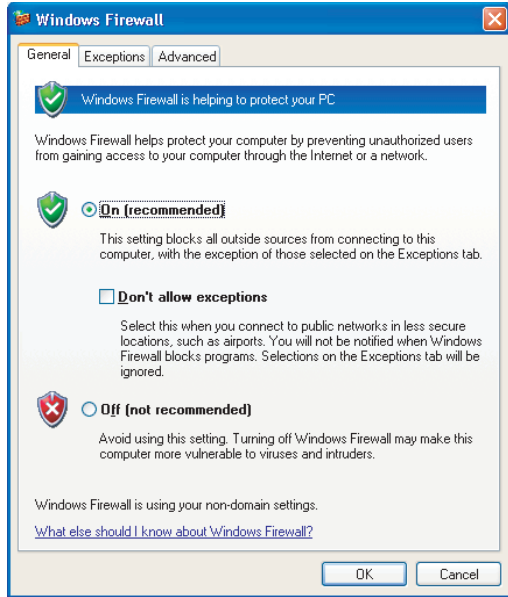
The firewall no longer blocks the ETAS product in question (in the example: ASCET). This decision survives a restart of the program, or even the PC.

Instead of waiting for the "Windows Security Alert" dialog window, you can unblock ETAS products in advance.

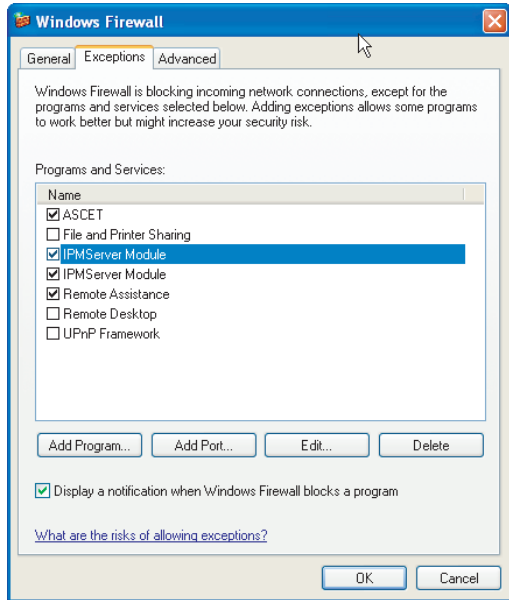
Unblocking ETAS products in the firewall control:

- From the Windows Start Menu, select **Settings → Control Panel**.

- In the control panel, double-click the **Windows Firewall** icon to open the "Windows Firewall" dialog window.

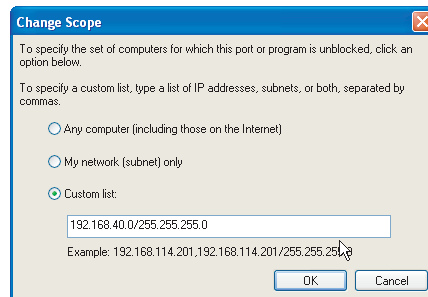


- In the "Windows Firewall" dialog window, open the "Exceptions" tab.



This tab lists the exceptions not blocked by the firewall. Use **Add Program** or **Edit** to add new programs, or edit existing ones.

- Make sure that the ETAS products and services you want to use are properly configured exceptions.
 - Open the "Change Setup" window.



- To ensure proper ETAS hardware access, make sure that at least the IP addresses 192.168.40.xxx are unblocked.
- Close the "Change Setup" window with **OK**.
- Close the "Windows Firewall" dialog window with **OK**.

The firewall no longer blocks the ETAS product in question (in the example: ASCET). This decision survives a restart of the PC.

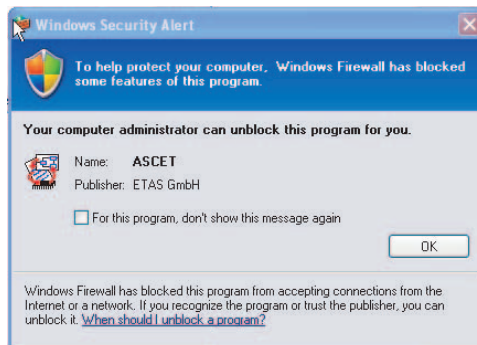
9.2 Users without Administrator Privileges

This section addresses users with restricted privileges, e.g., no system changes, write restrictions, local login.

Working with an ETAS product requires "Write" and "Modify" privileges within the ETAS, ETASData, and ETAS temporary directories. Otherwise, an error message opens if the product (e.g., ASCET) is started, and a database is opened. In that case, no correct operation of the ETAS product is possible because the database file and some *.ini files are modified during operation.

ASCET has to be installed by an administrator anyway. It is recommended that the administrator assures that the ETAS program/processes are added to the list of the Windows XP firewall exceptions, and selected in that list, after the installation. If this is omitted, the following will happen:

- The "Window Security Alert" window opens when one of the actions listed above (cf. page 227) is executed.



To unblock a program (no Admin privileges):

- In the "Windows Security Alert" dialog window, activate the option **For this program, don't show this message again**.
- Click **OK** to close the window.

An administrator has to select the respective product (e.g., ASCET) in the "Exceptions" tab of the "Windows Firewall" dialog window to avoid further problems regarding hardware access with that ETAS product.

9.3 Support and Problem Reporting

If you have any questions, contact the ETAS hotline.

	Phone	E-Mail
Europe (w/o France, Belgium, Luxembourg, Great Britain)	+49-711-89661-311 (ASCET) +49-711-89661-315 (INCA)	ec.hotline@etas.de inca.hotline@etas.de
France, Belgium, Luxembourg	+33-1-5670-0235 (ASCET) +33-1-5670-0234 (INCA)	support.ascet@etas.fr support.inca@etas.fr
Great Britain	+44-1283-546-512	support@etas-uk.net
Japan	+81-45-222-0951 (ASCET) +81-45-222-0950 (INCA)	ec.hotline@etas.co.jp inca.hotline@etas.co.jp
Korea	+82(2)5747-101 (ASCET) +82(2)5747-061 (INCA)	ec.hotline@etas.co.kr inca.hotline@etas.co.kr
USA	+1-888-ETASINC (382-7462)	support@etasinc.com

ETAS HQ

ETAS GmbH

Borsigstraße 14
70469 Stuttgart
Germany

Phone: +49 711 89661-0
Fax: +49 711 89661-105
E-mail: sales@etas.de
WWW: www.etasgroup.com

North America

ETAS Inc.

3021 Miller Road
Ann Arbor, MI 48103
USA

Phone: +1 888 ETAS INC
Fax: +1 734 997-9449
E-mail: sales@etas.us
WWW: www.etasgroup.com

Japan

ETAS K.K.

Queen's Tower C-17F
2-3-5, Minatomirai, Nishi-ku
Yokohama 220-6217
Japan

Phone: +81 45 222-0900
Fax: +81 45 222-0956
E-mail: sales@etas.co.jp
WWW: www.etasgroup.com

Great Britain

ETAS Ltd.

Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ
Great Britain

Phone: +44 1283 54 65 12
Fax: +44 1283 54 87 67
E-mail: sales@etas-uk.net
WWW: www.etasgroup.com

France

ETAS S.A.S.

1, place des Etats-Unis
SILIC 307
94588 Rungis Cedex
France

Phone: +33 1 56 70 00 50
Fax: +33 1 56 70 00 51
E-mail: sales@etas.fr
WWW: www.etasgroup.com

Korea

ETAS Korea Co. Ltd.

4F, 705 Bldg. 70-5
Yangjae-dong, Seocho-gu
Seoul 137-889
Korea

Phone: +82 2 57 47-016
Fax: +82 2 57 47-120
E-mail: sales@etas.co.kr

China

ETAS (Shanghai) Co., Ltd.

2404 Bank of China Tower
200 Yincheng Road Central
Shanghai 200120, P.R. China

Phone: +86 21 5037 2220
Fax: +86 21 5037 2221
E-mail: sales.cn@etasgroup.com
WWW: www.etasgroup.com

Index

A

- application mode 204
- ASAM-MCD-2MC file 205
- ASCET
 - firewall (Windows XP + SP2) 227
 - install basic system 18
 - path specifications 19
 - specify functional scope 21
 - specify license directory 23
 - store license file 41
 - structure 78
 - uninstall 34, 36

B

- Block Diagram Editor
 - buttons 84
- borrow license 44
- buttons
 - Block Diagram Editor 84
 - C Code Editor 87
 - Component Manager 83
 - CT Block Editor 87

- ESDL Editor 87
- Offline Experiment 91
- Project Editor 89

C

- C code 205
- C Code Editor
 - buttons 87
- calibration windows 205
- characteristic line 206
- characteristic map 206
- characteristic value 206
- class 206
- component 206
- Component Manager 206
 - buttons 83
- condition 206
- Configuration dialog box 206
- CT Block Editor
 - buttons 87

D

- data 207

- data generator 207
- data logger 207
- data set 207
- database 207
- default directories 218
- description file 207
- diagram 207
- dimension 207
- Distribution 207

E

- editor 207
 - element 235
- element 208
- element editor
 - open 235
- environment 208
- error
 - continue 216
 - exit 216
 - support function "Problem Report" 215
 - System Error window 215
 - what to do in case of ~ 216
- ESDL Editor
 - buttons 87
- event 208
- event generator 208
- experiment 208
- experiment environment 208

F

- fixed point code 208
- folder 208
- formula 208
- fullpass experiment 208

G

- General Operation
 - according to Windows conventions 93
 - drag & drop 95
 - function keys 219
 - hierarchy trees 95
 - monitor window 97
 - supporting functions 97
 - using the mouse 94

Glossary 203–213

H

- HEX file 209
- hierarchy 209

I

- icon 209
- implementation 209
- Installation
 - assign user privilege (Win 2000) 16
 - assign user privilege (Win XP) 16
 - cancel 25
 - customize configuration file 30
 - customize data for network ~ 32
 - install ASCET-MD 24
 - integrate modified database 33
 - integrate modified user profile 33
 - network installation 29
 - obtain license file 39
 - overwrite existing version 26
 - path specifications 19
 - specify ASCET functional scope 21
 - specify license directory 23
 - start ASCET installation 18
 - store license file 41
 - system requirements 15
 - uninstall ASCET 34, 36
 - without administrator privilege 29
- Intel Hex 209
- interface 209

K

- kind 210

L

- layout 210
- license
 - borrow 44
 - not found 41
 - return 45
 - return (normal case) 44
- license file
 - obtain 39
 - store 41

Licensing 39
 borrow license 43
 change borrowing time 43
 no license detected 41
 return borrowed license 44, 45
 show ~ status 42
 trial mode 41
literal 210

M

measure 210
measure window 210
measured signal 210
measurement 210
measuring channel parameters 210
message 210
methods 211
model type 211
module 211
monitor 211
Motorola-S-Record 211

O

obtain license file 39
Offline experiment
 buttons 91
Oscilloscope 211

P

parameter 212
priority 212
Problem Report 215
process 212
program 212
program description 212
project 212
project editor
 buttons 89

R

Reference Lists 215–225
resource 212
return borrowed license 44, 45

S

scheduling 212

scope 212
state 212
State machine 213
store license file 41
support function "Problem Report" 215

T

target 213
task 213
Transition 213
trial mode 41
type 213

U

user profile 213

V

variables 213

W

window elements 213

