ASCET V5.2

User's Guide

Copyright

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

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

© Copyright 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 EC010001 R5.2.2 EN

Contents

1 Introduction

All program parts and windows, the menu structure and how to operate the program are described in detail in the User's Guide. All menus and menu functions are explained. Before you work through this part of the manual, you should already be familiar with ASCET, i.e. you should already have worked through the chapter "Understanding ASCET" and the "Tutorial" in the ASCET Getting Started manual. The subsections are more or less in the order of the steps necessary for specifying components and projects.

All sections in the documentation follow the same principle. At the beginning of every main chapter, there is a short introduction, then a description of the user interface and menu items, followed by a workflow-oriented description of the individual steps.

1.1 Typical Workflow

The program structure of ASCET makes it necessary for the user to adapt some parts of his/her workflow to correspond to a specific scheme. There are several dialog sequences which cannot be changed. This results in a typical and efficient workflow which was taken into consideration in the documentation and is used as the guide. The following list is a rough overview of the order of the individual working steps:

- set up ASCET
- create the database and database entries
- add your own functions
- specify and simulate components
- specify and simulate projects
- create and edit datasets
- create and edit implementations
- use of signals and icons
- experimenting
- documentation of the results

1.2 Launching the Program

To launch ASCET:

- By default, the ASCET installation creates an icon on your desktop. Double-click this icon to launch the program.

- Alternatively, you can launch ASCET via Programs in the **Start** menu.

Below, there is a detailed description of the user interface of the Component Manager, with which ASCET is launched by default.

Note

You can determine software behavior individually. You can specify default paths, set different colors and fonts, make specifications for the import and export of data and lots more in ASCET. For more detailed information, please refer to chapter 2.2 "Component Manager – Setting Up ASCET".

2 The Component Manager

This chapter contains a detailed description of the Component Manager, its menu structure and operation. This window replaces both the ASCET start window and the database browser of previous versions.

The function of all menus and menu items is explained. Before reading this part, you should be familiar with ASCET by having studied chapter "Understanding ASCET" and the "Tutorial" in the ASCET "Getting Started" volume.

2.1 Component Manager – User Interface

2.1.1 General Description

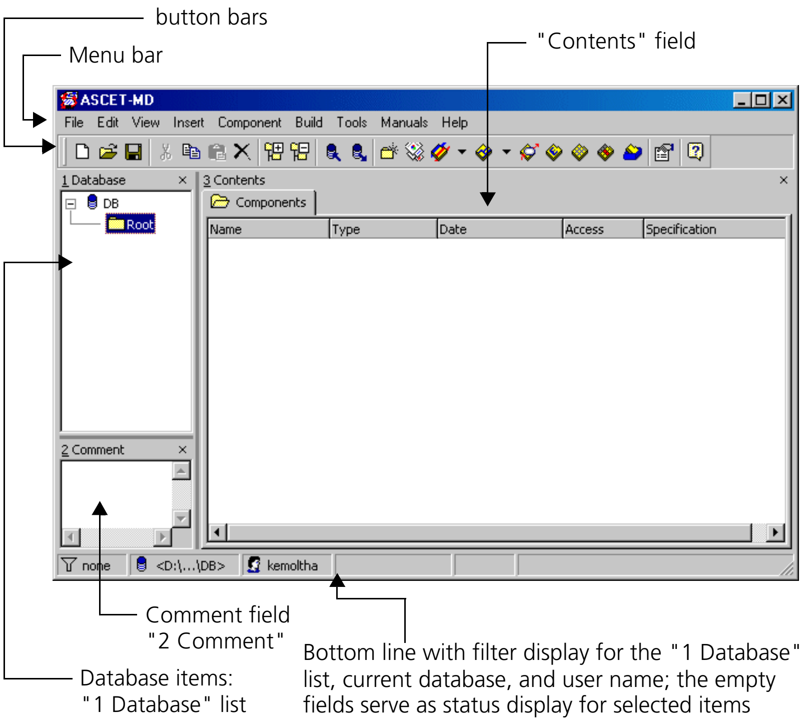
The ASCET Component Manager opens when the program starts. This is where you start the various editors (see chapter 4 "Specification of Components and Projects"). You can also manage several user profiles and, via an option dialog box, set station-, database-, and user-specific options, e.g., the definition of storage directories, preferences such as screen display options, or the start-up behavior of the software (see "Component Manager – Setting Up ASCET" on page 36).

The main purpose of the Component Manager is, however, to systematically store and manage all data created during the work with ASCET in a database (see "Component Manager – Managing Data" on page 73).

This section only describes the functions forming an integral part of the Component Manager. The editors, the experiment environment, as well as some add-ons, are described in separate chapters.

2.1.2 User Interface Component Manager

ASCET manages all the database items in a database in the Component Manager. You can create folders and subfolders, move, copy, import, and export individual items and also create entirely new databases. Besides organizing your data, you can also edit components and projects in ASCET. In earlier versions, this was only possible in the specification editors.


button bars
Menu bar
"Contents" field
ASCET-MD
File Edit View Insert Component Build Tools Manuals Help
1 Database
DB
Root
3 Contents
Components
Name
Type
Date
Access
Specification
2 Comment
none
<D:\...\DB>
kemoltha
Comment field "2 Comment"
Database items: "1 Database" list
Bottom line with filter display for the "1 Database" list, current database, and user name; the empty fields serve as status display for selected items

The folders and items contained in the current database are displayed in the "1 Database" list. The database name is the root of the tree structure. Each database contains one or more folders, which in turn contain other folders and database items. They are organized and displayed in a hierarchy in the "1 Database" list.

The "2 Comment" field is a text field that contains any notes or internal comments on the current folder or item. Typically, this field is used for developer comments and internal version details. This information is not included in the documentation generated automatically. It is not to be confused with the information entered via the notes editor (see chapter 7.4 "Notes" on page 693).

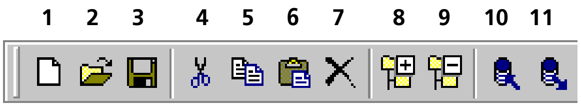
Note

*During export (cf. chapter 2.3.3 on page 89), the content of the "2 Comment" field is exported **only** for top-level folders and items. Comments on subfolders are **not** exported.*

The "3 Contents" field displays information about the selected folder or database item in various tabs. The information displayed in the "3 Contents" field varies depending on the selection (folder or database item) and the type of database item. Details are given in chapter 2.1.3 "Views in the Component Manager".

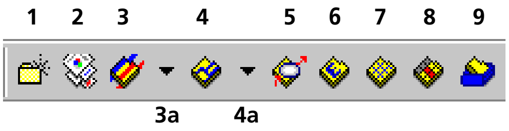
Description of the Control Elements in the Button Bars

Default Button Bar:



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

"Insert" Button Bar:

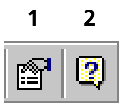


1. Insert Folder
2. Insert Project
3. Insert Module - <Type>
4. Insert Class - <Type>

   The arrows (3a and 4a) can be used to select the object type - block diagram, C code, or ESDL.
5. Insert State Machine

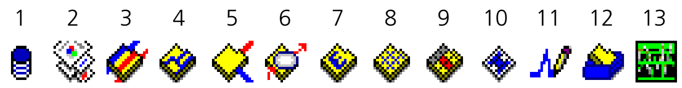
6. Insert Container
7. Insert Boolean Table
8. Insert Conditional Table
9. Insert Container

"Tools" Button Bar:



1. Options
2. ? (opens the "AboutASCET" window with information on the installed products of the ASCET product family)

Description of the Symbols for the Database Items



1. database
2. project
3. module
4. class
5. CT block
6. state machine
7. enumeration
8. Boolean table
9. conditional table
10. icon
11. signal
12. container
13. ASAM-MCD-2MC project

- **File**
  - *New Database (<CTRL> + <N>)*

    Create a new database.
  - *Open Database (<CTRL> + <O>)*

    Open database.
  - *Close Database*

    Close database.
  - *Save Database (<CTRL> + <S>)*

    Saves the changes in the database.
  - *Save Database as...*

    Creates a backup copy of the entire database at a location you specify.
  - *Import (<CTRL> + <M>)*

    Imports database entries from an export file and shows them in the "1 Database" list.
  - *Import* directory

    Imports the entire content of a directory.
  - *Export (<CTRL> + <E>)*

    Note

    This function is only available when ASCET-MD is installed. In that case, it is also available as a context menu in the "1 Database" list.

    Exports selected items or entire folders from the "1 Database" list into one or more export files (`*.exp`).
  - *1 D:\..\DB*
    (and other, similar entries)

    List of most recently opened databases.
  - *Exit (<ALT> + <F4>)*

    exit ASCET

- **Edit**

Note

*The functions **Cut**, **Copy**, **Paste**, **Delete** and **Rename** are also available as context menus in the "1 Database" list.*
*Except **Find** and **Query**, these functions are only available if ASCET-MD is installed.*

  – *Cut (<CTRL> + <X>)*

    Cuts the selected database entry from the database.

  – *Copy (<CTRL> + <C>)*

    Copies the selected database entry to the clipboard.

  – *Paste (<CTRL> + <V>)*

    Pastes a database item from the clipboard to the selected folder.

  – *Delete (<DEL>)*

    Finally deletes the selected database item. When a folder is deleted, all items in the folder are deleted as well.

    Note

    *Be careful when using the **Edit** → **Delete** function; it **cannot** be undone.*

  – *Rename (<F2>)*

    Renames the selected database item.

  – *Find (<CTRL> + <F>)*

    Searches a string in C code or ESDL components.

  – *Replace (<CTRL> + <H>)*

    Replaces a string in C code or ESDL components.

  – *Query (<CTRL> + <Q>)*

    Searches the database from various points of view (see section "Browsing the Database" on page 133).

- **View**

  – *Expand all*

    Displays the whole database content (folders and database items).

  – *Collapse all*

    Only the database name is displayed.

- *Filter* → *<component type>*

  Provides filters for the database items in the "1 Database" list.

- *Show /Hide*

  Shows/hides several window elements.
  Database List → the "1 Database" list,
  Comment → the "2 Comment" field,
  Contents → the "3 Contents" field,
  Monitor → the monitor window

- *Update (<*F5*>)*

  Updates the Component Manager.

• **Insert**

Note

*The **Insert** menu is also available as context menu in the "1 Database" list. It is only available when ASCET-MD is installed.*

- *Folder (<*INSERT*>)*

  Adds a new folder.

- *Project*

  Adds a new project.

- *Module*

  Adds a new module;
  Block Diagram → as block diagram,
  ESDL → in ESDL,
  C Code → in C code.

- *Class*

  Adds a new class (submenus as for modules)

- *Continuous Time Block*

  Adds a new CT block (submenus as for modules)

- *State Machine*

  Adds a new state machine.

- *Enumeration*

  Adds a new enumeration.

- *Boolean Table*

  Adds a new Boolean Table.

- *Conditional Table*

  Adds a new conditional table (s. chapter 4.7).
- *Icon*

  Adds a new icon
- *Signal*

  Adds a new signal.
- *Container*

  Adds a new container.

• **Component**

Note

The ***Edit Item*** *function is also available as context menu in the "1 Database" list.*
Edit Layout*,* ***Replace References****,* ***Become another Item*** *and* ***Reproduce as*** *are only available when ASCET-MD is installed.*

- *Edit Item (*<RETURN>*)*

  Opens the editor for the selected database item.
- *Edit Layout*

  Edit a component layout.
- *Enumeration*

  only available when an enumeration is selected in the "1 Database" list, and the focus is on the "3 Contents" field (see page 28).
- *Disallow Import*

  Disallows overwriting of database items during import.
- *Access Rights*

  Changes the access rights of a folder or item.
- *Password*

  Activates and deactivates password protection (only for folders).
- *Show References*

  Displays the references to a component.
- *Replace References*

  Replace the references to a database item.

- *Become another Item*

  Replacement of a database item.

- *Reproduce As*

  Copies the structure of a component
  Block Diagram → to a block diagram,
  ESDL → in ESDL,
  C Code → in C code.

- *Notes*

  Edits the notes for a database item.

- *Reset Operator Implementation*

  Resets the operator implementations
  Flat → in the selected components,
  Recursive → in the selected components and the referenced coponents.

• **Build**

- *Touch All*

  All database components are marked as *changed*. Thus, a compilation of the entire project is enforced.

- *Clean All*

  The code of the project stored in the database is discarded.

• **Tools**

- *Documentation → Contents*

  Opens the "Documentation Contents" window. This is where the items to be documented are selected.

- *Documentation → Options*

  Setting options for documentation generation.

- *Database → Performance Utilities*

  Opens the "Database Info for: <*Database*>" window. It provides four, partly combinable, database administration functions.

  Optimize is used to recombine the various scattered fragments in the database that result from intensive editing (defragmentation).

  Convert is used to cleanup the internal references between the data records ("X uses Y") after changes have been made to the database structure, thus improving access speeds.

Repair is used to rebuild the database from scratch using the "Optimize" and "Convert" functions.

Check is used to verify the structures and references in the database while logging the results in the monitor window.

– *Database* → *Convert*

  Converts the database;

  Modify components to force impl/data update → all components are set to *modified* state, so that the next time a component is accessed, it's implementation is checked and—if necessary—adjusted;

  All Names to ANSI-C → all names in the database are converted to ANSI-C;

  System Constants to Constants → converts system constants to constants (cf. chapter "The Kind of Elements" in the ASCET reference guide);

  Variables to Volatile → assigns the *volatile* attribute to all variables in the database (cf. page 112 and page 451);

  Variables to Nonvolatile → assigns the *non-volatile* attribute to all variables in the database;

  Components/Elements to default memory location → inserts the memory location Default into all element implementations in the database;

  Operator Implementations to Impl. Casts → replaces existing operator implementations with implementation casts (cf. page 508);

  Reset Operator Implementations → removes all operator implementations from the database).

– *Database* → *List Operator Implementations*

  Displays a list of operator implementations in the ASCET monitor window.

– *Database* → *Compare Database*

  Compares the current database with a selected second database.

– *Arithmetic Service Editor*

  Opens the editor for arithmetic services for a target (see chapter 4.14);

  PC → PC target;
  more submenus are added when ASCET-RP or ASCET-SE are installed.

- *Network Settings*

  Opens the "Network settings for ETAS hardware (Page 1)" window; only relevant for ASCET-RP.

- *Views*

  Opens the "Views" window, where views are managed (cf. chapter 7.3 on page 683).

- *Options*

  Opens the "Options" window, which allows the setting of various options (cf. chapter 2.2 on page 36).

• **Manuals**

Note

If you did not install the manuals, the menu contains no functions.

- *English user's guide*

  Opens the English user's guide in the Acrobat Reader.

- *English reference*

  Opens the English reference manual in the Acrobat Reader.

- *German user's guide*

  Opens the German user's guide in the Acrobat Reader.

- *German reference*

  Opens the German reference manual in the Acrobat Reader.

- *Open manuals folder*

  Opens the `ETAS\ETASManuals` directory in the Windows-Explorer.

When you install ASCET-RP or ASCET-SE, further menu functions for the respective manuals are added.

• **Help**

- *Contents*

  Opens the content of the ASCET online help.

- *Index*

  Opens the index of the ASCET online help.

- *Loaded Packages*

  Displays a list of all installed integration modules (see chapter "System Information" in the ASCET "Getting Started" manual) in the monitor window.

- *Loaded Targets*

  Displays a list of the installed ASCET-SE in the monitor window.

- *Product Disclaimer*

  Opens the disclaimer window containing a danger notice.

- *Problem Report*

  Starts the Problem Report" tool (see chapter 2.1.4).

- *Hotkey Assignment*

  Opens a window listing the keyboard shortcuts.

- *About*

  Opens a window with information on the installed products of the ASCET product family.

- *Support*

  Opens a window with the ASCET hotline addresses.

- *License Info*

  Opens a window with licensing information (see chapter "Licensing" in the ASCET "Getting Started" manual).

Context-Sensitive Menu Options

The options of the **Edit** and **Component** menus described here have various meanings, depending on the focus. This section lists the meaning of the menu functions for each tab in the "3 Contents" field, provided the focus is on that field.

Note

*The options of the two menus **not** described in this section either have the same meaning regardless of the focus or are deactivated.*

"Element" Tab:

- **Edit**
  - *Copy (<CTRL> + <C>)*

    Copies the selected elements to the database clipboard.

  - *Paste (*<CTRL> + <V>*)*

    Copies the elements from the database clipboard to the component.

  - *Delete* (<Del>)

    Deletes the selected elements from the component.

  - *Rename (*<F2>*)*

    Renames the selected element.

- **Component**

  - *Edit Item (*<RETURN>*)*

    Opens the element editor for the selected element.

"Data" Tab:

- **Edit**

  - *Copy (*<CTRL> + <C>*)*

    Copies the current data of the selected element to the database clipboard.

  - *Paste (*<CTRL> + <V>*)*

    Pastes the data from the database clipboard to the selected element.

  - *Delete* (<Del>)

    Deletes the selected elements from the component.

  - *Rename (*<F2>*)*

    Renames the selected element.

- **Component**

  - *Edit Item (*<RETURN>*)*

    Opens the data editor for the selected element.

"Implementation" Tab:

- **Edit**

  - *Copy (*<CTRL> + <C>*)*

    Copies the current implementation of the selected element to the database clipboard.

  - *Paste (*<CTRL> + <V>*)*

    Pastes the implementation from the database clipboard to the selected element.

  - *Delete* (<Del>)

    Deletes the selected elements from the component.
  - *Rename* (<F2>)

    Renames the selected element.
- **Component**
  - *Edit Item* (<RETURN>)

    Opens the implementation editor for the selected element.

An enumeration is selected in the "1 Database" field:

- **Edit**
  - *Delete* (<Del>)

    Deletes the selected enumerator from the component.
  - *Rename* (<F2>)

    Renames the selected enumerator.
- **Component**
  - *Enumeration → Add* (<INSERT>)

    Adds an enumerator.
  - *Enumeration → Shift Up* (<CTRL> + <U>)

    Moves an enumerator upward in the list.
  - *Enumeration → Shift Down* (<CTRL> + <D>)

    Moves an enumerator downward in the list.

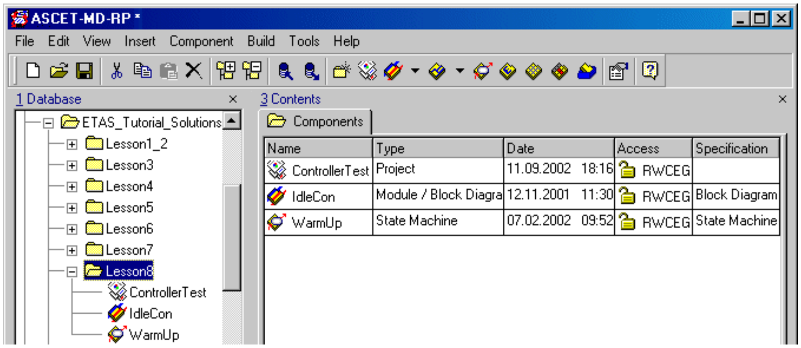
2.1.3 Views in the Component Manager

ASCET offers an enhanced concept for viewing the database items. Folder, components, projects, containers and enumerations can be displayed in the "3 Contents" field under various aspects. For components and projects, the tabs of the "3 Contents" field offers an element, data, and implementation view; for components, a layout view is offered, too. For folders, the field offers the folder view, which displays the objects in the folder as well as some information about the objects. The container view is much the same as the folder view.

Each tab offers several editing possibilities which are available via the **Edit** and **Component** menus or the context menu in the "3 Contents" field.

To select the folder view.

- Highlight a folder in the "1 Database" field.



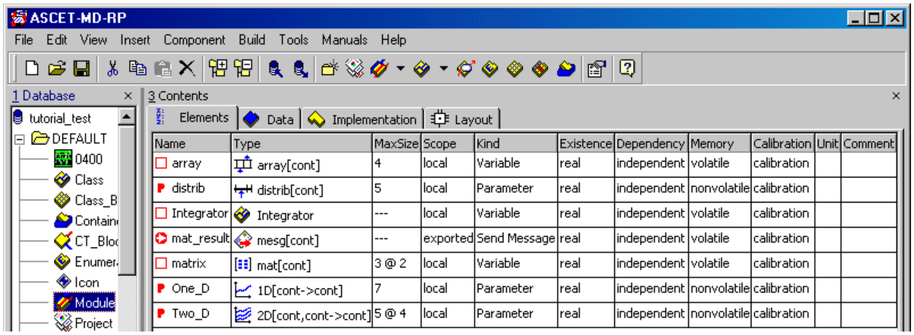
Name	Type	Date	Access	Specification
ControllerTest	Project	11.09.2002 18:16	RWCEG	
IdleCon	Module / Block Diagra	12.11.2001 11:30	RWCEG	Block Diagram
WarmUp	State Machine	07.02.2002 09:52	RWCEG	State Machine

The content of the folder, as well as the name, type, creation date and time, access rights and method of component creation are displayed.

Section "Editing Components and Projects in the Component Manager" on page 82 explains how to work in this tab.

To select the Element view of a component or project.

- Select a component or project in the "1 Database" field.
- In the "3 Contents" field, click on the "Elements" tab to display the element view.



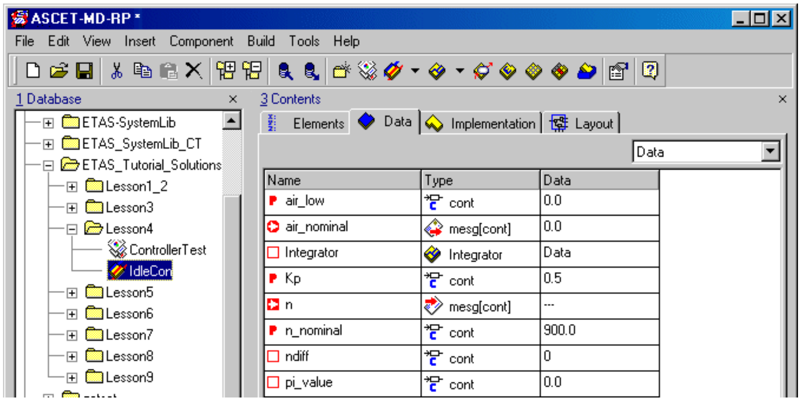
Name	Type	MaxSize	Scope	Kind	Existence	Dependency	Memory	Calibration	Unit	Comment
array	array[cont]	4	local	Variable	real	independent	volatile	calibration		
distrib	distrib[cont]	5	local	Parameter	real	independent	nonvolatile	calibration		
Integrator	Integrator	---	local	Variable	real	independent	volatile	calibration		
mat_result	mesg[cont]	---	exported	Send Message	real	independent	volatile	calibration		
matrix	mat[cont]	3 @ 2	local	Variable	real	independent	volatile	calibration		
One_D	1D[cont->cont]	7	local	Parameter	real	independent	nonvolatile	calibration		
Two_D	2D[cont,cont->cont]	5 @ 4	local	Parameter	real	independent	nonvolatile	calibration		

All elements of the component or project, and the name, type, scope, type and unit of the elements, are displayed. If a comment was entered for an element (cf. page 451), it is shown here, too.
For arrays, matrices and characteristic lines/ maps, the maximal size is shown, too.

Section "Editing Components and Projects in the Component Manager" on page 82 explains how to edit elements.

To select the Data view:

- Select a component or project in the "1 Database" field.
- In the "3 Contents" field, click on the "Data" tab for the Data view.



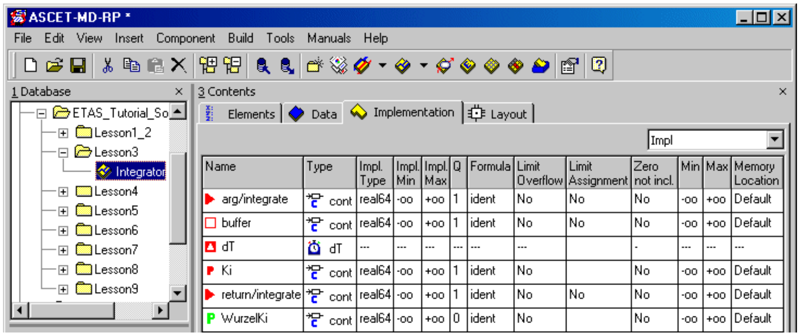
Name	Type	Data
air_low	cont	0.0
air_nominal	mesg[cont]	0.0
Integrator	Integrator	Data
Kp	cont	0.5
n	mesg[cont]	---
n_nominal	cont	900.0
ndiff	cont	0
pi_value	cont	0.0

Name, type and data of the elements are shown. Use the combo box to select another dataset.
Section "Editing Components and Projects in the Component Manager" on page 82 explains how to edit the data.

To select the implementation view:

- In the "1 Database" field, select a component or project.

- In the "3 Contents" field, click on the "Implementation" tab to display the implementation view.



Name	Type	Impl. Type	Impl. Min	Impl. Max	Q	Formula	Limit Overflow	Limit Assignment	Zero not incl.	Min	Max	Memory Location
arg/integrate	cont	real64	-oo	+oo	1	ident	No	No	No	-oo	+oo	Default
buffer	cont	real64	-oo	+oo	1	ident	No	No	No	-oo	+oo	Default
dT	dT	---	---	---	---	---	---		-	---	---	---
Ki	cont	real64	-oo	+oo	1	ident	No		No	-oo	+oo	Default
return/integrate	cont	real64	-oo	+oo	1	ident	No	No	No	-oo	+oo	Default
WurzelKi	cont	real64	-oo	+oo	0	ident	No		No	-oo	+oo	Default

All the information about the current implementation is displayed. Use the combo box to select another implementation.
Section "Editing Components and Projects in the Component Manager" on page 82 explains how to edit the implementations.

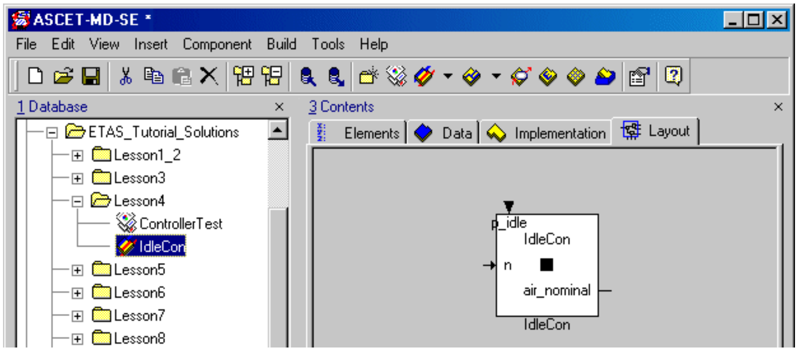
To select the Layout view of a component:

- In the "1 Database" field, select a component.

Note

Projects have no layout.

- In the "3 Contents" field, click on the "Layout" tab to display the layout view.

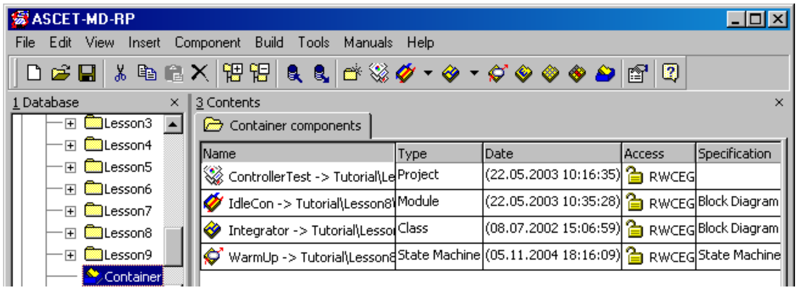


The component layout is displayed.

Section "Editing Components and Projects in the Component Manager" on page 82 explains how to edit the layout.

To select the container view:

- In the "1 Database" field, select a container.



Name	Type	Date	Access	Specification
ControllerTest -> Tutorial\Le	Project	(22.05.2003 10:16:35)	RWCEG	
IdleCon -> Tutorial\Lesson8\	Module	(22.05.2003 10:35:28)	RWCEG	Block Diagram
Integrator -> Tutorial\Lesso	Class	(08.07.2002 15:06:59)	RWCEG	Block Diagram
WarmUp -> Tutorial\Lesson8	State Machine	(05.11.2004 18:16:09)	RWCEG	State Machine

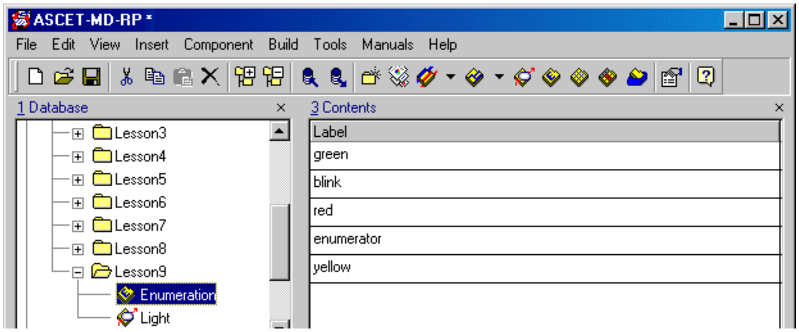
The content of the container, as well as the name, type, creation date and time, access rights and creation method of the objects inside are displayed.

Section 4.9.1 "Working with Containers" explains how to work in this tab.

To select the enumeration view:

- In the "1 Database" field., select an enumeration.

  The enumerators are listed in the "3 Contents" field.



Label
green
blink
red
enumerator
yellow

Section "To create an enumeration:" on page 76 explains how to edit enumerations.

2.1.4 ETAS "Problem Report" Support Function

ASCET offers you a support function to inform ETAS about problems that occurred during your work with the program. 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 `EtasLogFiles01.zip` in the `...\ETAS\LogFiles\` directory. For additional archive files, the file name is incremented automatically (up to `19`) to avoid immediate overwriting older archive files.

Note

A maximum of 20 archive files can be created; if more archive files are used, older files are overwritten starting at `00`.

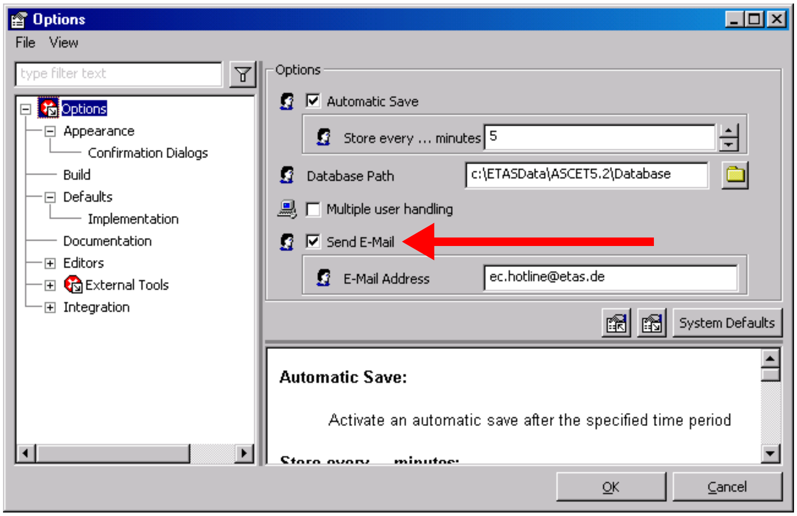
You can set up ASCET so that this archive is automatically sent to the ETAS hotline service. For this purpose, you need a MAPI-compliant E-mail program (e.g., MS Exchange or Eudora). For other e-mail programs, you need to send the archive yourself as an attachment.

To set up the "Problem Report" support function:

- In the Component Manager, select the **Tools** → **Options** menu option.

  The "Options" window opens.

- In the "Options" node, activate the **Send E-Mail** option.



Options

File View

type filter text

Options
Appearance
Confirmation Dialogs
Build
Defaults
Implementation
Documentation
Editors
External Tools
Integration

Options

☑ Automatic Save

Store every ... minutes 5

Database Path c:\ETASData\ASCET5.2\Database

☐ Multiple user handling

☑ Send E-Mail

E-Mail Address ec.hotline@etas.de

System Defaults

Automatic Save:

Activate an automatic save after the specified time period

Store every ... minutes:

OK Cancel

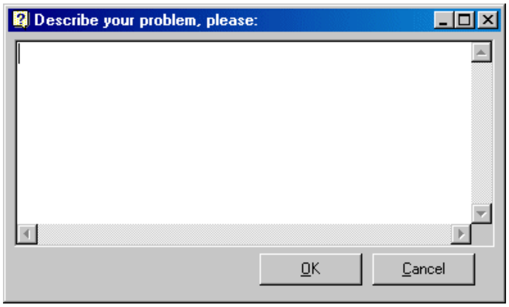
Thus, the automatic transmisson of the e-mail to the ETAS hotline is enabled.
If your mail program is not MAPI-compliant, this function is ignored.

- If necessary, change the address of the hotline service in the "E-Mail Address" field. The current address is:
  `ec.hotline@etas.de`

To send a problem report:

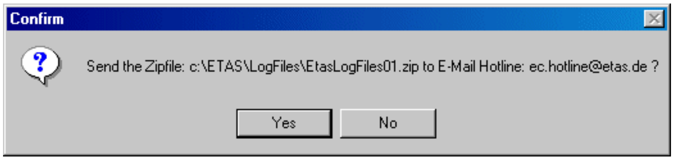
- In the Component Manager, select **Help** → **Problem Report** to send your problem to ETAS.

  A dialog window opens and prompts for a description of the problem.



- Type your description and click **OK**.

  Your license is checked. When you are using a licensed ASCET version, the report is compressed into an archive file. When automatic transmission is activated, the archive is sent immediately. Otherwise, you are asked whether the archive is to be sent.



Send the Zipfile: c:\ETAS\LogFiles\EtasLogFiles01.zip to E-Mail Hotline: ec.hotline@etas.de ?

- Click **Yes** to send the e-mail; otherwise, click **No**.

  If your mail program is not MAPI-compliant, you have to send the archive file manually, e.g., as an attachment.

2.2 Component Manager – Setting Up ASCET

This chapter describes the configuration options of ASCET and how to work with user profiles.

Note

This manual describes the ASCET-MD options. Add-on products, such as ASCET-RP, can add their own options; these are described in the respective add-on program.

You can set various options in ASCET. There are two types of options: general and user-specific options. Both types are managed in the "Options" window and stored in XML files. When ASCET is booted for the first time after installation, these XML files do not exist. They are created automatically and assigned the default settings of the system or—if an older ASCET version was installed—the settings of the older version.

General options (icon) are specific to the ASCET installation on your workstation. They comprise the target directory for code generation, several paths, code preview settings, and the selection of single- or multi-user mode for running the program.

General options are stored in the data directory in the `stationSettings.xml` file.

User-specific options (icon) comprise a variety of topics, ranging from settings for diagrams, export/import file paths to miscellaneous general options which are described in more detail later in this chapter. With these options, you can create a user environment customized to meet your particular needs. Changes to the user options are stored separately for each user if user selection is activated on startup. New user profiles start with the default settings.

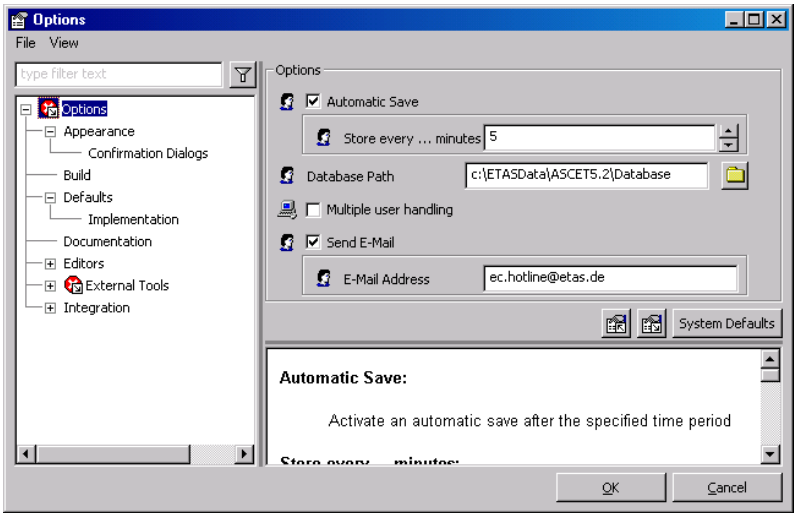
User-specific options are stored in the directory of the relevant user (`ETAS-Data\ASCET5.2\User\`<*username*>) in the `userSettings.xml` file.

Note

Before setting user-specific options, you should decide whether your ASCET installation is to run in single- or multi-user mode to make sure your options are stored in the correct user profile.

Options with invalid values are indicated by a red circle containing a white X: .

User interface of the ASCET options window. The ASCET options window contains the following elements:



Options

☑ Automatic Save

Store every ... minutes: 5

Database Path: c:\ETASData\ASCET5.2\Database

☐ Multiple user handling

☑ Send E-Mail

E-Mail Address: ec.hotline@etas.de

Automatic Save:

Activate an automatic save after the specified time period

Store every ... minutes:

- at the top left, a filter for the options
- on the left, a tree view with the option groups as nodes

  The ⊗ or ⊗ symbol next to a node indicates that the node, or one of its subnodes, contains an invalid option.
- at the top right, the display field for the options of the selected node
- at the bottom right, a short description of the options displayed
- the buttons **Import Options for selected Node from XML File**, **Export Options of selected Node into XML File** and **System Defaults**
- the **File** menu with the following menu options
  - **Import**

    Imports the options for the selected node from an XML file.
  - **Export**

    Exports the options of the selected node to an XML file.
- the **View** menu with the following menu options
  - **Expand All**

    The tree view is expanded as far as possible.

- **Collapse All**

  The tree view is collapsed as far as possible.
- **Show/Hide Description**

  Shows/hides the brief description of the options displayed.

To set ASCET options:

- In the Component Manager, select **Tools** → **Options**

or

- click the **Options** button.

  The ASCET options window ("Options" dialog window) opens.

  The options are ordered by topic; each topic is represented as a node in the tree view on the left in the "Options" window.
- Select the required node to make the settings for ASCET.

  Some options appear as radio buttons, input fields or combo boxes. Other options show a button next to their names which opens another dialog window.
- Make the new setting as appropriate for the respective control element.

  Options which have been changed are indicated by an *; options with a setting other than the default one are displayed in bold.

  You can set options successively in different nodes of the "Options" window.
- Click the **System Defaults** button to restore the default settings for the current node.

- When you have made all settings, click on **OK**.

  The settings are applied and the "Options" window closed.

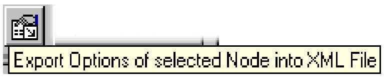
Note

Some new settings may not take effect immediately. For example, you modify the list font for the ASCET user interface; this does not affect any windows that are already open. You must close the relevant window and reopen it for the new settings to take effect.

To export options:

Once you have set the options of a node, you can export them to an XML file. Open the ASCET options window.

- Select the node whose options you want to export.
- If necessary, set the options of this node.


Export Options of selected Node into XML File

- Click the **Export Options of selected Node into XML File** button

or

- select **File** → **Export**.

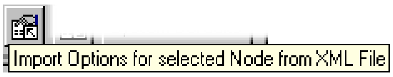
  The Windows file selection window opens. `*.xml` is specified as format.
- Set path and name of the export file.
- Click **Save**.

  The options of the selected node are written to the XML file specified.

To import options:

Exported options can be reimported again. This means that, for example, identical options can be defined for several users.

- Open the ASCET options window.
- Select the node whose options you want to import.


Import Options for selected Node from XML File

- Click the **Import Options for selected Node from XML File** button

or

- select **File** → **Import**.

  A confirmation dialog opens.

- Activate the option **Remember my Decision** if you want to give the same answer to *all* questions of this type (see page 45).

  In that case, the "Import Options" window no longer opens. You can revoke this setting in the ASCET options window, "Confirmation Dialogs" node (see "Options for Confirmation Dialogs" on page 45).

- Confirm the overwriting of the existing options with **OK**.

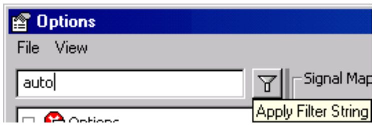
  The Windows file selection dialog box opens. `*.xml` is specified as format.

- Select the XML file which contains the options required.
- Click **Open**.

  The options contained in the XML file specified are applied to the options window.

To filter options:

You can filter the options to make the display more concise.



- Enter a text in the filter text field.
- You can enter one or mor words, or a part of a wort.
- Click the **Apply Filter String** button.

  The display field shows only options whose name, value, or description, contains the filter text.
  The tree view shows only nodes than contain options matching the filter. If necessary, further nodes are displayed to maintain an intact hierarchy.

2.2.1 General Options

You set a few general options in the top node of the options window, the "Options" node (see figure on page 38).

Option	Value	Description
Automatic Save	Activated/ deactivated	Activates/deactivates the automatic saving function.
Store every ... minutes	Number	Time interval for automatic saving
Database Path	Path	Main ASCET database directory
Multiple user handling	Activated/ deactivated	Activates/deactivates user selection.
Send E-Mail	Activated/ deactivated	Activates/deactivates automatic e-mail transfer.
E-Mail Address	Address	Hotline e-mail address

With the **Automatic Save** option, you can activate the automatic saving function and enter the time interval you want. Your data will then be saved according to the intervals you entered.

For performance reasons, some user operations requiring an entry in the database are only held in a cache. Using **Automatic Save**, you can force the database cache to be saved in cycles.

To set up automatic saving:

- Activate the **Automatic Save** option.

  Automatic saving is switched on.

- Enter a value in "Store every ... minutes"

or

- use the arrow keys to adjust the time interval between two updates.

  The database cache is saved regularly at the specified interval.

The **Multiple user handling** option specifies whether your ASCET installation should handle user profiles. For more details, refer to chapter 2.2.11 "Working with User Profiles".

The **Send E-Mail** option allows you to enable the automatic transmission of e-mails to the ETAS hotline.

To enable the automatic transmission of e-mails:

- Activate the **Send E-Mail** option.
- In the "E-Mail Address" field, enter the current hotline address.
- Click **OK** to accept the setting.

  The next time you select **Help** → **Problem Report**, an e-mail is automatically sent to the hotline. This function is described in detail in chapter 2.1.4 on page 33.

2.2.2 Appearance Options

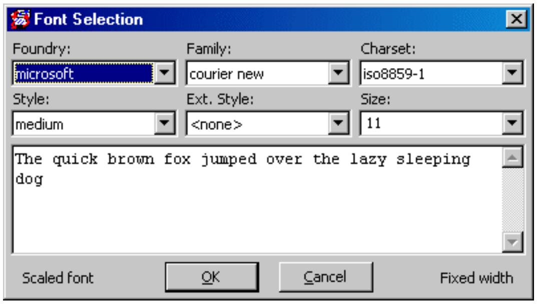
In the "Appearance" node, you set the options which influence the appearance of ASCET or of individual editors. Options of the same type are compiled in subnodes.

Option	Value	Description
Extended Infos in Database Browser	Activated/ deactivated	Activates/deactivates the display of information on access rights in the Component Manager.
Show Disclaimer	Activated/ deactivated	Activates/deactivates the display of the disclaimer when ASCET is launched.
List Font	Font	font for list entries
Text Font	Font	Font for text on the user interface
Edit Font Mapping	Directives	Mapping Windows fonts to Postscript fonts

To set fonts:

- Click on the button next to the font you want to change.

  The "Font Selection" window opens.



Foundry: microsoft
Family: courier new
Charset: iso8859-1
Style: medium
Ext. Style: <none>
Size: 11

```
The quick brown fox jumped over the lazy sleeping
dog
```

Scaled font

Fixed width

- In the "Foundry" combo box, select the origin of the font.
- In the "Family" combo box, select the font.
- In the "Charset" combo box, select the character set.
- In the "Style" combo box, select style and inclination of the font.
- In the "Size" combo box, select the font size.

  The available sizes depend on the selected font.

  The text field below the combo boxes shows a sample sentence in the currently selected font. In the "Ext. Style" combo box, as well as to the left and right of the **OK** and **Cancel** buttons, further information on the selected font is given.
- Click **OK** to accept the selection and close the "Font Selection" window.

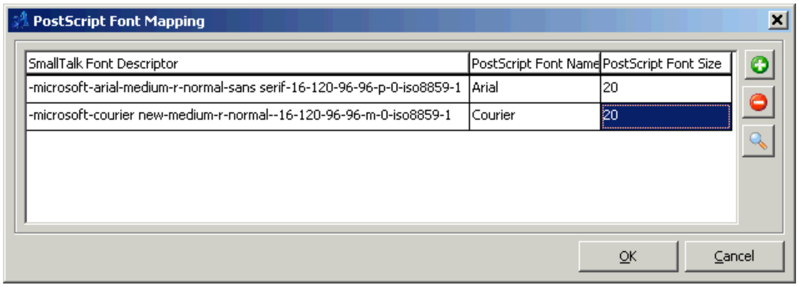
  The font selection does not affect existing windows.

To map fonts:

With the "Edit Font Mapping" option, you can map the Windows fonts to Postscript fonts. When a block diagram is saved as a Postscript file, the Windows fonts are replaced by the selected Postscript fonts.

- Click the button in the "PostScript" subnode.

  The "PostScript Font Mapping" window opens. It shows a table of existing mapping entries for fonts.



PostScript Font Mapping

SmallTalk Font Descriptor	PostScript Font Name	PostScript Font Size
-microsoft-arial-medium-r-normal-sans serif-16-120-96-96-p-0-iso8859-1	Arial	20
-microsoft-courier new-medium-r-normal--16-120-96-96-m-0-iso8859-1	Courier	20

Column	Meaning
`SmallTalk Font Descriptor`	Windows font (SmallTalk)
`PostScript Font Name`	PostScript font
`PostScript Font Size`	Size of PostScript font

1. Adding/removing mapping entries

- Click the **Add new Mapping Entry** button in the "PostScript Font Mapping" window to create a new mapping entry.

Note

You can map a Windows font to several Postscript fonts. When generating Postscript files, only the first mapping entry for a font is used; the others have no effect.

- Click the **Remove selected Mapping Entry** button in the "PostScript Font Mapping" window to remove a mapping entry.

2. Editing mapping entries

- Double-click in a cell of the "SmallTalk Font Descriptor" column in the "PostScript Font Mapping" window.

  The "Font Selection" window (see page 43) opens.
- Set the Windows font you want to map to a Postscript font in the "Font Selection" window and click **OK**.
- Double-click a cell in the column "PostScript Font Name" in the "PostScript Font Mapping" window and enter the Postscript font which is to replace the Windows font.
- Double-click a cell in the column "PostScript Font Size" in the "PostScript Font Mapping" window and enter the size for the Postscript font.
- Click the **Preview selected Mapping Entry** button in the "PostScript Font Mapping" window to display the Postscript command generated with the selected settings.

3. Concluding work

- Close the "PostScript Font Mapping" window with **OK** to accept the settings.
- Close the "PostScript Font Mapping" window with **Cancel** to reject the settings.

Options for Confirmation Dialogs

When working with ASCET, confirmation windows are displayed at various points. You can hide these windows, if you like. In the "Confirmation Dialogs" node of the "Options" window, you can show hidden confirmation windows (☒ symbol). You can also hide confirmation windows here.

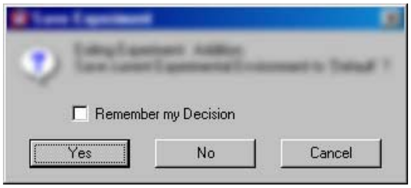
The main purpose of this node is to show confirmation windows you have hidden while working with ASCET. It is *explicitly not recommended* to use the node to hide confirmation windows you have not yet seen in the program.

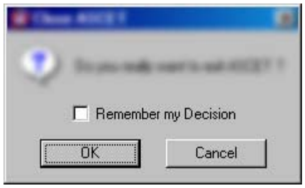
"Dialog" column	"Decision" column	Description
Change DB Path	`Show` / `Yes` / `No`	Confirmation window for changing the database path when opening a database which is not in the default directory.
Close ASCET	`Show` / `OK`	Confirmation window when ending ASCET.
Close Boolean Table	`Show` / `OK`	Confirmation window when closing a Boolean table with wrong entries.
Close existing Editor	`Show` / `OK`	Confirmation window when an editor is opened again for the same component.
Deassign Application Mode <inactive>	`Show` / `OK`	Confirmation window when the type of an init task with application mode `inactive` is changed in the OS editor.
Edit complex Element	`Show` / `Hide`	Determines whether the element editor for complex elements opens automatically (`Show`) or not (`Hide`).
Edit CT Element	`Show` / `Hide`	Determines whether the element editor for CT blocks opens automatically (`Show`) or not (`Hide`).
Edit Implementation Cast	`Show` / `Hide`	Determines whether the element editor is opened (`Show`) or not (`Hide`) when generating a new implementation cast.
Edit primitive Element	`Show` / `Hide`	Determines whether the element editor should be opened (`Show`) or not (`Hide`) when generating a new element (variable, parameter, characteristic curve/map).
Export Components	`Show` / `OK`	Confirmation window during export with activated **One File for Each Item** option.
Export Message Configuration	`Show` / `OK`	Confirmation window when exporting message configurations (see chapter 2.4.2)
Import Options	`Show` / `OK`	Confirmation window when importing options.
Import Views	`Show` / `OK`	Confirmation window when importing views.
Overwrite Files/Folders	`Show` / `OK`	Confirmation window when importing existing components.

"Dialog" column	**"Decision" column**	**Description**
Resize Drawing Area	`Show` / `Yes` / `No`	Confirmation window when, upon opening a block diagram, the size of the drawing area is about to be changed.
Save Experiment	`Show` / `Yes` / `No`	Confirmation window for saving an experiment when the experiment is quit.
Set Layout to Default	`Show` / `OK`	Confirmation window when the layout of a new component is set as default layout (see page 233).
Set Options to Default	`Show` / `OK`	Confirmation window when resetting options.
Set Target Settings to Default	`Show` / `OK`	Confirmation window when a project with unknown target or compiler is opened in ASCET.
Start Offline Simulation	`Show` / `OK`	Confirmation window when an offline experiment is started and the event generator contains no event.
Store Changes in Graphic	`Show` / `Yes` / `No`	Confirmation window when a block diagram with unsaved changes is closed.

The possible values of an option depend on the confirmation dialog window.

- `Show` is always available; the confirmation window is displayed if `Show` is selected.
- `Yes` and `No` are available if the confirmation window contains **Yes** and **No** buttons.
- `OK` is available if the confirmation dialog contains an **OK** button.



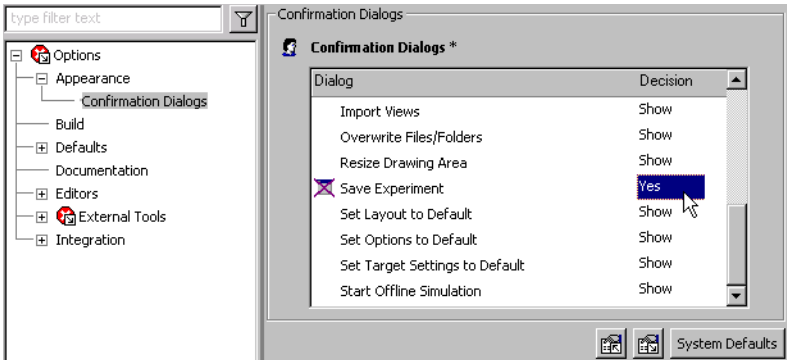


`Yes`, `No`, and `OK` hide the window; further effects correspond to the button of the same name.

- `Hide` is available for those options that control automatic opening of the element editor for new elements. `Hide` prevents automatic opening of the element editor.

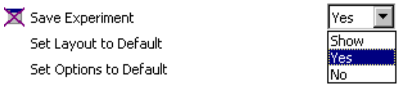
To show/hide confirmation dialog windows:

- In the ASCET options window, open the "Confirmation Dialogs" node.
- In the "Coonfirmation Dialogs" field, double-click the entry in the "Decision" column next to the desired option.



Dialog	Decision
Import Views	Show
Overwrite Files/Folders	Show
Resize Drawing Area	Show
Save Experiment	Yes
Set Layout to Default	Show
Set Options to Default	Show
Set Target Settings to Default	Show
Start Offline Simulation	Show

- Select an entry from the combo box.



From now on, the confirmation window is treated according to the selection.

2.2.3 Build Options

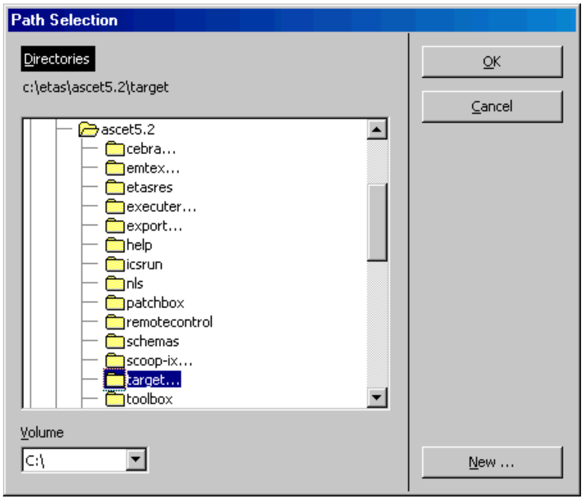
In the "Build" node, you set options which influence code generation in ASCET.

Option	Value	Description
Code Preview Path	Path	Directory for the generated C-code or HTML/XML files
Code Generation Path	Path	Target directory for code generation; this is where files generated during code generation are stored. Default: `ETAS\Ascet5.2\cgen`
Target Root Path	Path	Target main directory; this is where subdirectories for the different targets are created.
Specific Path	Path	Settable path for project files
Write Project Files on Build	Activated/ deactivated	Determines whether project files are written to the hard disk during code generation.
Keep files in Code Generation Directory	Activated/ deactivated	Specifies whether the code generation directory ("Code Generation Path" option) is to be retained or deleted when exiting ASCET.

To set a path:

- If you want to change a path, click on the button next to the path option you want to change.

  The "Path Selection" window opens.



- If necessary, select a volume in the "Volume" combo box.
- Select an existing directory from the "Directories" list

or

- create a new directory using the **New** button.
- Click **OK**.

  The selected directory is displayed in the options window.

2.2.4 Default Options

In the "Defaults" node, you can make default settings for newly created projects (also default projects) as well as for implementations. The default settings for implementations are compiled in the "Implementation" subnode.

Option	Value	Description
Use Arithmetic Service	Activated/ deactivated	Activates/deactivates the use of arithmetic services for new projects.
Use first available Service set	Activated/ deactivated	Assigns new projects the first set of arithmetic services in the file `services.ini`.
Use Project Options Template	Activated/ deactivated	Activates/deactivates the use of an XML file as template for project settings of new projects.
Project Options Template	File	Name and path of the XML file which is used as template for the project settings (only available if **Use Project Options Template** is activated).

Default setting for arithmetic services:

To activate arithmetic services for newly created projects, proceed as follows.

- Activate the option **Use Arithmetic Service** in the "Defaults" node.

  This activates the use of arithmetic services for newly created projects.

- Activate the option **Use first available Service Set**.

  A newly created project uses the first set of arithmetic services of the `services.ini` file.

- Click **OK** to accept the setting.

For more details on arithmetic services, refer to chapter 4.14.

Option	Value	Description
Limit to maximum bit length	Activated/deactivated	Specifies whether the result of an operation is to be limited in the case of an overflow.
Implementation Master	`Model` / `Implementation`	Implementation master
Resolution Handling	`Automatic` / `Reduce Resolution` / `Keep Resolution`	Specifies how the resolution is to be handled in the case of an overflow.
Minimum cont Data Type	`real64`/ `real32`/ `uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Minimum type for implementations of `cont` variables
Minimum log Data Type	`bit`/ `uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Minimum type for implementations of `log` variables
Minimum sdisc Data Type	`uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Minimum type for implementations of `sdisc` variables
Minimum udisc Data Type	`uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Minimum type for implementations of `udisc` variables
Default cont Data Type	`real64`/ `real32`/ `uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Default type for implementations of `cont` variables
Default log Data Type	`bit`/ `uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Default type for implementations of `log` variables
Default sdisc Data Type	`uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Default type for implementations of `sdisc` variables
Default udisc Data Type	`uint8`/ `int8`/ `uint16`/ `int16`/ `uint32`/ `int32`	Default type for implementations of `udisc` variables

To set up default options for implementations:

- Activate the option **Limit to maximum bit length** if the result of an operation is to be limited in the case of an overflow.
- In the "Implementation Master" combo box, select the master page to determine newly created implementations.

  When you create a new implementation, the selected page is used automatically as master page in the implementation editor.
- In the "Minimum <*type*> Data Type" combo box, select a minimum implementation data type for `<type>` variables.
- In the "Default <*type*> Data Type" combo box, select a default implementation data type for `<type>` variables.

  `<type>` can be `cont`, `log`, `sdisc` or `udisc`.

  The selected types are preset when you edit the implementations of the respective variables.

  When you open a database from an older ASCET version, `real64` is selected for `cont` variables, `int8` for `sdisc` variables, `uint8` for `udisc` variables and `int8` for `log` variables.
- Click **OK** to accept the setting.

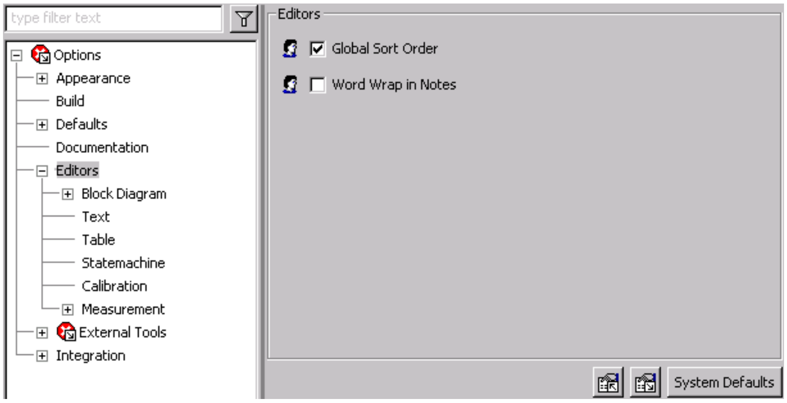
2.2.5 Options for Automatic Documentation

In the "Documentation" node, you can make global settings for automatic documentation (for more details see chapter 7).

Option	Value	Description
Documentor Working Path	Path	Working directory for documentation: Target directory for the documentation generated.
Scale EPS Graphic	Activated/ deactivated	Specifies whether the EPS graphic generated with **View** → **Save as Postscript** is scaled to the paper size (see page 57).

2.2.6 Options for Editors

In the "Editors" node, you can make settings for different editors, as well as for measure and calibration windows. Settings which belong to just one editor are compiled in subnodes.



☑ Global Sort Order

☐ Word Wrap in Notes

Option	Value	Description
Global Sort Order	Activated/ deactivated	Activates/deactivates the use of a global sort order in the element list of all editors.
Word Wrap in Notes	Activated/ deactivated	Activates/deactivates line break in the notes editor

Options for Block Diagrams

The options in the "Block Diagram" node refer to all block diagrams.

Option	Value	Description
Activate flexible layout	Activated / deactivated	Determines whether the layout of an included component can be changed in the block diagram editor or not.
Size of Undo Buffer	Number	Number of working steps in the undo buffer of block diagram and state machine editor
Drawing Area Size (px)	`2000@2000` / `5000@5000` / `10000@10000`	Size in pixel of the drawing area for block diagrams.
Selection Marker Size (Pixel)	Number	Selection marker size in block diagrams

Option	Value	Description
Show Watermark	Activated / deactivated	Determines whether the name of the selected view is displayed as a water-mark in the block diagram.
Watermark Size (px)	Number	Font size of the watermark
Show Page Number	Activated / deactivated	Shows the page number in block dia-grams.
BDE Graphical Comments	Font	Font for user comments in block dia-grams
BDE Graphical Names	Font	Font for names of graphical elements in block diagrams

To activate a flexible layout (global):

- In the "Block Diagram" node, activate the option **Activate flexible layout**.

  Now you can change the layout of included components in the block diagram or in the "Graphics" tab of the project editor.

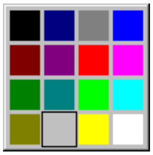
Setting fonts is described on page 43.

"Colors" node: The color settings for block diagrams are compiled in the options window in the "Colors" node.

Option	Value	Description
Background Color	Color	Background color in the block diagram edi-tor and the project editor, "Graphics" tab
Grid Color	Color	Color of the grid in block diagrams
Comment Line Color	Color	Color of comment lines in block diagrams or projects
Selection Marker Color	Color	Color of the selection markers in block dia-grams
Watermark Color	Color	Color for the watermark, i.e. the name of the selected view, as well as for page num-bers and print-page markers.

To specify colors:

You can determine various color settings for block diagrams as well as the input area (inputs of Boolean tables, see chapter 4.6, and the condition area of conditional tables, see chapter 4.7) and output area (outputs of Boolean tables and action area of conditional tables).



- Select the combo box of the option you want to change.

  A color selection prompter is displayed. The current color is indicated.
- Select the desired color.

  For the color selection to become effective, close and re-open an existing window.

"Sequencing" node: This node compiles options for sequence calls (see chapter 4.1.6).

Option	Value	Description
Sequence Shift Offset	Number	Offset for the shift of sequence call numbers
Sequence Step Size	Number	Step size for the automatic scaling of sequence call numbers
Use gaps	Activated/ deactivated	Fills the gaps between existing sequence numbers with automatic numbering of sequence calls.

To set sequencing options:

- In the "Sequence Step Size" field, enter the step size for the automatic scaling of sequence call numbers.
- In the "Sequence Shift Offset" field, enter the value for the automatic shift of sequence call numbers.
- Activate the **Use gaps** option when you want the automatic numbering of sequence calls to fill the gaps between existing sequence numbers.

"Paper Size" node: This node contains the options which control the printing of block diagrams.

Option	Value	Description
Paper Size	`A5 (148x210)/` `A4 (210x297)/` `Letter (216x279)/` `Legal (216x355)/` `Userdefined`	Determines the size of the print area. If `Userdefined` is selected, the size can be adjusted using the following two options.
User Size Width (mm)	Number	User-defined print area: Width
User Size Height (mm)	Number	User-defined print area: Height
Paper Orientation	`Landscape` / `Portrait`	Orientation of the print area.

Options for Text Editors

The options in the "Text" node concern the C-Code and ESDL editor, but not the state or transition editor.

Option	Value	Description
Tabulator Size	Number	Size of the tabulator in the text editors
Code Font	Font	Font for text in ESDL and C-Code editors

Setting fonts is described on page 43.

Options for Table Editors

The options in the "Tables" node affect the editors for Boolean tables and conditional tables.

Option	Value	Description
Input Color	Color	Background color for the input area of a table
Output Color	Color	Background color for the output area of a table

Setting colors is described on page 56.

Options for State Machines

In the "Statemachine" node, you determine default settings for state machines.

Option	Value	Description
Use ESDL as default for state machine	Activated/ deactivated	Default for actions of states and transitions
Animated States Color	Color	Color of animated states in experiments

To set default options for state machines:

- Activate the **Use ESDL as default for state machine** option.
- Click **OK** to accept the setting.

  If you create a new state or transition in a state machine, ESDL is used as default for the transition and state editors.

Setting colors is described on page 56.

Options for Calibration Windows

In the "Calibration" node, you determine default settings for calibration windows. These settings, and others, can be changed in the setup for individual calibration windows.

For the meaning of the options, refer to the descriptions in the Options window.

Options for Measurement Windows

In the "Measurement" node, you determine default settings for calibration windows. These settings, and others, can be changed in the setup for individual calibration windows.

For the meaning of the options, refer to the descriptions in the Options window.

"Datalogger" node: This node contains options for the data logger (see chapter 6.1.9).

Option	Value	Description
Auto Increment Data-Logger File Name	Activated/ deactivated	Determines whether the log file of the data logger is numbered consecutively (activated) or overwritten at each recrding.

2.2.7 Experiment Options

The "Experiment" node contains options for offline and online experiments.

Option	Value	Description
Initialize variables at OS start	Activated/ deactivated	Determines whether variables are initialized at each start of the simulation (offline experiment) or the operating system (online experiment).
Initialize parameters at OS start	Activated/ deactivated	Determines whether parameters are initialized at each start of the simulation (offline experiment) or the operating system (online experiment).

2.2.8 Options for External Tools

The "External Tools" node contains options for external tools. Some external tools have separate subnodes, e.g., the compilers or the text editor which can be called from the ESDL or C code editor or from the ASCET monitor window. Other options are set directly in the "External Tools" node.

Option	Value	Description
Perl Compiler	File	Perl compiler
External Make Utility	File	External make file
Output Redirector Utility	File	Redirects output
C Preprocessor	File	C-code preprocessor
C Code Beautifier	File	Reformats the generated code in accordance with user specifications

To set the paths for external tools:

- In the "External Tools" node, click the button next to the option you want to change.

  The Windows file selection dialog box appears.

- Select the directory that contains the tool.
- Select the required file and click **Open**.

  File name and path are shown to the right of the button.

ASCII Editor Options

Option	Value	Description
Use external ASCII Editor	Activated/ deactivated	Activates/deactivates the use of an external text editor.
ASCII Editor	File	Name and path of the external text editor
Commandline Arguments	characters	Command line arguments for the external text editor; first argument: file name, seond argument: line number
Begin RegionMarker	characters	Marker for the text editor to identify the beginning of a region.
End RegionMarker	characters	Marker for the text editor to identify the end of a region.

The combo boxes for command line arguments and region markers offer pre-defined values for some text editors.

Compiler Options

The "Compiler" node contains subnodes for the existing compilers. By default, these are Borland-C V4.5, Borland-C V5.5 and Microsoft Visual C++. If you install ASCET-RP or ASCET-SE, there are additional compilers.

The options—except the last one—are available for all compilers.

Option	Value	Description
Tool Root Path	Path	Installation directory of the compiler
Private directive	Directive	Compiler directive for private functions
Public directive	Directive	Compiler directive for public functions
Unique identifier	String	Unique identifier for the compiler
Object file extension	File exten-sion	File extension for object files
Result file extension	File exten-sion	File extension for executables
Supports long filenames	Activated/ deactivated	Indicates whether the compiler supports long file names.

Option	Value	Description
Inline directive	Directive	Compiler directive for inline functions
Change CYGWIN Registry Settings		*GNU-C V2.95.3 (PowerPC) only*; changes the Cygwin registry settings to guarantee corrrect function with the GNU compiler.
Supports precompiled header	Activated/ deactivated	Cannot be edited. Shows whether the compiler supports precompiled headers.

2.2.9 Options for Integration

The "Integration" node contains options for import/export, data exchange etc. Related options are compiled in subnodes.

Option	Value	Description
Tool Server Activation	Activated/ deactivated	For internal use; should always be activated.
Use .NET based Tool API	Activated/ deactivated	Toggles between the current .NET-based Tool API and the (outdated) MS J++ Tool API.

Export Options

The "Export" node allows you to specify in which way and to what extent the database objects selected will be exported. In this way, you can store individual datasets as separate files and specify that the referenced entries will be exported as well.

You can also specify whether a description file is created when database items are exported. Format and content of the description file can be determined too.

Option	Value	Description
Default Export Path	Path	Default export directory
Include Referenced Items	Activated/ deactivated	Activates/deactivates recursive export.
One File for each Item	Activated/ deactivated	Activates/deactivates export of items in separate files.
Write ASCII file on Binary Export	Activated/ deactivated	Activates/deactivates the generation of additional ASCII description files.
Write XML file on Binary Export	Activated/ deactivated	Activates/deactivates the generation of additional XML description files.

Option	Value	Description
Create HTML File	Activated/ deactivated	Generates an HTML file for the XML description file.
Write Default Project Info	Activated/ deactivated	Adds information about the default project to the XML description file.
Write Version Info	Activated/ deactivated	Adds information on the version to the XML description file.
Write Graphic Specification	Activated/ deactivated	Adds graphic information to the XML description file.
Sort XML Tags	Activated/ deactivated	Activates/deactivates alphabetic sorting of the tags in the XML description file.
Create complete Hierarchy	Activated/ deactivated	Specifies whether the exported files are stored in a directory structure which reflects the folder structure of the ASCET database during AMD export.
Export Experiment Environment	Activated/ deactivated	Specifies whether the experiment environment is exported together with the component.
Encryption Key	Key	Key for file encryption during AMD export

To set export options:

- Select the default target directory for export in the "Default Export Path" field.
- Activate the **Include Referenced Items** option to export database items recursively (i.e. including all referenced items).
- Activate the **One File for each Item** option to write each exported database item in a separate file.
- Activate the **Write ASCII file for Binary Export** option if you want to create an additional ASCII description file.
- Activate the **Write XML file for Binary Export** option if you want to create an additional XML description file.

  The options in the field under **Write XML file for Binary Export** are now available.

  - Activate or deactivate the options to determine the content of the XML description file.

- Activate or deactivate the options in the "AMD Format" field to determine the content of the AMD description file.

For information on exporting database items, see chapter 2.3.3 "Exporting Folders and Database Items" on page 89.

Import Options

In the "Import" node, you set options for importing database entries. Options for automatic repair during AMD import are compiled in the "Autofixes" node (page 64).

Option	Value	Description
Default Import Path	Path	Default import directory
Discard Existing Implementation	Activated/ deactivated	Activates/deactivates replacement of all existing implementations with the imported implementations.
Keep Folder Path in Database	Activated/ deactivated	Activates/deactivates the usage of the path set in the database when you import existing components.
Remove Version Information	Activated/ deactivated	Activates/deactivates the removal of information from a version management tool.
Keep Hierarchy	Activated/ deactivated	Specifies whether the directory structure of the export files in the ASCET database is reflected during AMD import.
Import referenced Items	Activated/ deactivated	Activates/deactivates recursive AMD import.
Remap OIDs	Activated/ deactivated	Assigns new object IDs to all imported components. Thus, copies of existing items are created.
Decryption Key	Key	Key for file decryption during AMD import

To set import options:

- Enter the default directory for imports in the "Default Import Path" field.

- Activate the **Discard Existing Implementations** option to replace *all* existing implementations with the imported implementations.

  All previously existing implementations are lost; only those of the imported component are retained.

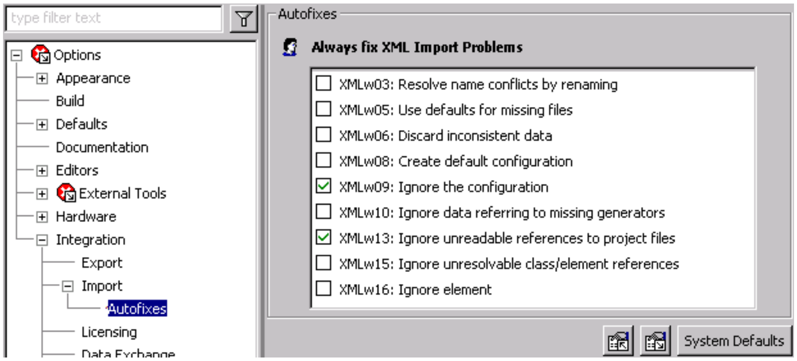
  This option is only relevant if an existing component is imported.
- Activate the **Keep Folder Path** option to keep the path set in the database.

  If you deactivate this option, the path stored in the export file is used for the import of existing components.
- Activate the **Remove Version Information** option to remove existing version information.

  If you import a versioned item without activating the option, you will not be able to store it in your own version management database unless you have access to the original archive. If in doubt, activate the option.
- Use the options in the "AMD Format" field to set up the AMD import.

For information on importing database items, see "Importing Folders and Database Items" on page 93.

"Autofixes" Node: This node contains solutions for possible problems during AMD import (see "Special Features of the AMD Import"). You can select which problems are solved automatically, without notification.



Autofixes

Always fix XML Import Problems

- [ ] XMLw03: Resolve name conflicts by renaming
- [ ] XMLw05: Use defaults for missing files
- [ ] XMLw06: Discard inconsistent data
- [ ] XMLw08: Create default configuration
- [✓] XMLw09: Ignore the configuration
- [ ] XMLw10: Ignore data referring to missing generators
- [✓] XMLw13: Ignore unreadable references to project files
- [ ] XMLw15: Ignore unresolvable class/element references
- [ ] XMLw16: Ignore element

Licensing Options

In the "Licensing" nod, you

Option	Value	Description
Borrow license for days	Number	Length of time a license can be borrowed (see ASCET Getting Started manual, chapter "Licensing").
Activate idle time shutdown	Activated/ deactivated	Determines whether ASCET shuts down after an idle period or not.
Idle period [min]	Number	Determines the length of the idle period in minutes.

Data Exchange Options

In the "Data Exchange" node, you set the options for data exchange. The exchange formats DCM V1.x and 2.x and data exchange with INCA and ASCET from V4 are supported.

Option	Value	Description
Data File Path	Path	Directory of the data file
Extension for Output File	`dcf` / `dcm` / `kon`	File extension for the exchange file
DCM Format version	`DCM V1.x` / `DCM V2.x`	Determines the DCM version to be used.
Write Enums	Activated/ deactivated	Determines whether enumerations are written in TEXT format (only for `DCM V1.x`).
Write Sampling Points	Activated/ deactivated	Determines whether sampling points are written in DCM V2.x syntax (only for `DCM V1.x`).
Case Sensitive Names	Activated/ deactivated	Specifies whether upper/lower case are taken into account. If the option is disabled, e.g. a parameter named `CONT` is mapped to a parameter named `Cont` during import.
Hierarchical Names	Activated/ deactivated	Switches between complete (activated) and simple item names of global items.
Include Booleans	Activated/ deactivated	Specifies whether Booleans are taken into account.

Option	Value	Description
Boolean format	`true`/ `false` / `Integer`	Determines the format of Booleans (only for DCM V2.x).
Show Log File for Load / Show Log File for Save	Activated/ deactivated	Specifies whether the log file for read/ write processes is displayed.
Log File for Load/ Log File for Save	File name	Path and name of the log files for read and write processes

Executable File Options

The setting options for the executable files are located in the "HexFile" node.

Option	Value	Description
HexFile Format	`IntelHex` / `MotorolaSRecord`	Default format for exporting a hex file
IntelHex Record Size	`16` / `32` / `64`	Permissible number of bytes per field for the IntelHex format
SRecord Count	Activated/ deactivated	Determines whether the number of data fields is written at the end of each block (and before a possible subsequent termination record).
SRecord Format	`16` / `24` / `32`	Address width in bits in Motorola format
SRecord Size	`16` / `24` / `32`	Permissible number of bytes per field for the Motorola format
SRecord Termination	Activated/ deactivated	Specifies whether a termination record is put at the end of each block or not (and before a possible subsequent termination record).

The options **SRecord*** are only of any importance if `MotorolaSRecord` was selected under "HexFile Format".

2.2.10 External Options

It is possible to include user-defined options. This takes place using an XML file which you modify to suit your own requirements. The file is stored in the ASCET installation directory; for ASCET to recognize it, it must have the extension `*`**`.aod.xml`**.

The definition of an option has the following basic format (code sections in italics must be replaced with suitable values for each option):

```
<OptionDeclaration
```

```
        xmlCategory="path"
        optionCategory="value"
        optionClass="type"
        attributeName="option name"
        optionFile="filename.xml">
        <Group>path</Group>
        <Label>text</Label>
        <Description>text</Description>
        <Tooltip>text</Tooltip>
        <InitialValue>value</InitialValue>
        <DefaultValue>value</DefaultValue>
    </OptionDeclaration>
```

Some option types have additional items; these are described in Tab. 2-2.

The meaning of the attributes and items is listed in Tab. 2-1.

Attribute/Item	Meaning
`xmlCategory`[a]	Path under which the option in the XML file from `optionFile` is stored, e.g. `Sample\Option`.
`optionCategory`	Way the option is saved (`FILE` – in a file, `FIXED` – not saved).
`optionClass`	Type of option; for possible values see Tab. 2-2.
`attributeName`	Name of the option in the XML file from `optionFile`. Has to be unique in the file.
`optionFile`[a]	XML file in which the value of the option is stored.
`<Group>`	Path (`Node\Subnode\...`) of the option in the ASCET options window; either new or existing.
`<Label>`	Name of the option in the ASCET options window.
`<Description>`	Short description of the option.
`<Tooltip>`	Tooltip text of the option in the ASCET options window.
`<InitialValue>`	Initial value; is only used if no value from `optionFile` is saved in the XML file.
`<DefaultValue>`	Default value; is used for **System Defaults** and as an initial value if `<InitialValue>` is not set.

a: The values of several options can be stored in the same XML file and/or under the same path.

Tab. 2-1 Meaning of the Attributes and Items

The following types are available for the definition of user-defined options:

`optionClass`	**Explanation**
`EtasBooleanOption`	Option box
`EtasButtonOption`	Button that executes a command

	`<InitialValue>` and `<DefaultValue>` are set to the value `ignored`.
	Additional items for the definition:
	`  <ButtonOption>`
	`    <Action>`*`path/executable file`*`↵`
	`                    </Action>`
	`  </ButtonOption>`

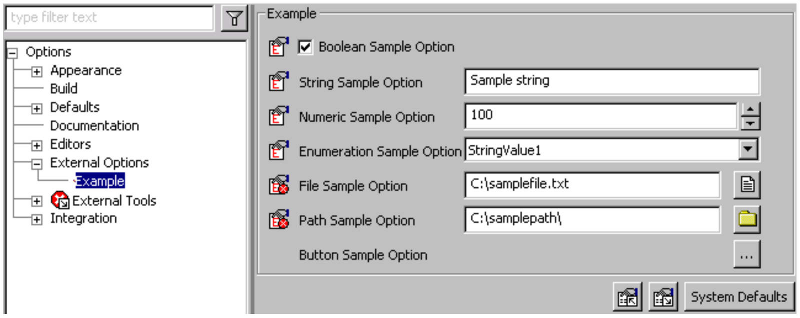
`EtasEnumerationOption`	Combo box for selecting a character string

	Additional items for the definition:
	`  <EnumerationOption>`
	`    <StringValues>`
	`      <StringValue>`*`string1`*`      ↵`
	`                 </StringValue>`
	`      ...`
	`      <StringValue>`*`stringN`*`      ↵`
	`                 </StringValue>`
	`    </StringValues>`
	`    <Values>`
	`      <Value>`*`value1`*`</Value>`
	`      ...`
	`      <Value>`*`valueN`*`</Value>`
	`    </Values>`
	`  </EnumerationOption>`

`optionClass`	**Explanation**
`EtasFileOption`	Text field for path and name of a file as well as a button for file selector window
	Additional items for the definition: `<FileOption>` `<DialogTitle>`*`text`* ↵ `</DialogTitle>` `<SearchPath>`*`path`*`</SearchPath>` `<SearchMask>`*`mask`*`</SearchMask>` `<InvalidCharacters>`*`characters`* `</InvalidCharacters>` `<FilterTypes>` `<FilterType `*`filter`*`>` `<Description>`*`text`* ↵ `</Description>` `</FilterType>` `...` `</FilterTypes>` `</FileOption>`
`EtasFloatOption`	Entry field with arrow buttons for float values
	Additional items for the definition: `<FloatOption>` `<MinValue>`*`value`*`</MinValue>` `<MaxValue>`*`value`*`</MaxValue>` `</FloatOption>`
`EtasNumericOption`	Entry field with arrow buttons for integer values
	Additional items for the definition: `<NumericOption>` `<MinValue>`*`value`*`</MinValue>` `<MaxValue>`*`value`*`</MaxValue>` `</NumericOption>`
`EtasPathOption`	Text field for directory path with button for "Path Selection" window
`EtasStringOption`	Text field for entering a character string

Tab. 2-2 Types of User-Defined Options (code sections in italics must be replaced with suitable values for each option)

Your installation CD contains the sample file `externalOptionsExample.aod.xml`. This file defines the options shown below.



type filter text

- Options
  - Appearance
  - Build
  - Defaults
  - Documentation
  - Editors
  - External Options
    - Example
  - External Tools
  - Integration

Example

☑ Boolean Sample Option	
String Sample Option	Sample string
Numeric Sample Option	100
Enumeration Sample Option	StringValue1
File Sample Option	C:\samplefile.txt
Path Sample Option	C:\samplepath\
Button Sample Option	...

System Defaults

2.2.11 Working with User Profiles

You can create and manage several user profiles in ASCET. This function enables the selection of specific settings in ASCET which apply to one user only. If, for example, several users share a computer, each user can save an ASCET configuration customized to meet his/her requirements in his/her personal user environment. This includes options such as screen font, font size, several settings for the various editors, default settings for elements, etc.

Note

Even if only one user uses a computer, (s)he can set up several user profiles. For example, (s)he can create an optimized user profile for working in the office and one for travelling.

All settings in a respective user profile are global and apply irrespective of the concrete task.

During program execution, only the user profile for the current user environment can be changed. Editing user profiles is only possible when the functionality has been activated. Use the **Tools** → **Options** function ("Options" node, see chapter 2.2.1 on page 41) to determine whether editing user profiles is possible during program execution. Another user profile can only be selected by restarting ASCET.

Using the User Selection Feature

By activating user selection, the user can specify whether the dialog window for user profile selection is displayed.

If user selection is not activated, the program automatically uses the user profile for the user logged onto the system.

If user selection is activated, the user is prompted to log on the next time the program starts. The user can do this by selecting an existing user profile or adding a new profile to ASCET. This is described in the following.

To activate user selection:

- In the Component Manager, select **Tools** → **Options**

or

- click on the **Options** button.

  The "Options" window opens.
- In the "Options" node, activate the **Multiple user handling** option.

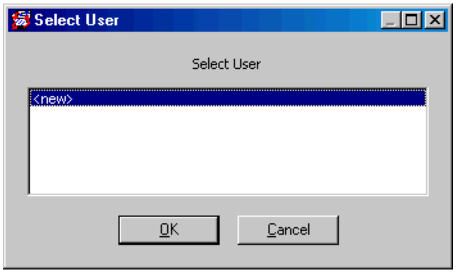
  The next time you start ASCET, you are prompted to either select an existing user name or to enter a new name.

To add a new user profile:

- Start ASCET with user selection activated in the "Options" window.

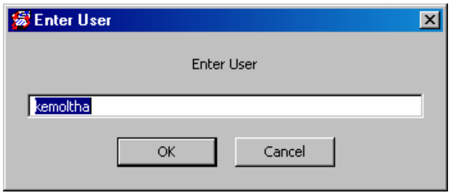
  The "Select User" window opens. It contains all available users.

  When user selection is first activated, the user list only contains one entry: `<new>`.



- Select the <new> entry from the list and click **OK**.

  The "Enter User" dialog window prompts for the name of the new user profile.



- Enter a name and click **OK**.

  A user profile name can consist of up to eight characters.

  ASCET is now started using a copy of the default user profile (default settings); the Component Manager opens. The name of the current user profile is displayed in the bottom bar of the window.

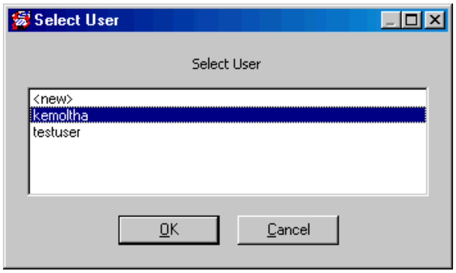
The user options specified during an ASCET session are stored in the user profile you select when the program starts. They become applicable again when the corresponding user name is selected at startup.

For instructions on how to customize the user environment to meet your requirements, read sections 2.2.1 – 2.2.10.

To activate an existing user profile:

- Start ASCET with user selection activated in the "Options" window.

  The "Select User" window opens.



- Select the name you want from the list and click **OK**.

  The Component Manager opens. The user options for the selected profile are activated.

Note

You cannot switch user profiles while the program is running. You must exit the current session and restart the program to change users.

2.3 Component Manager – Managing Data

As mentioned above, the main purpose of the Component Manager is to systematically store all data that is created during the work with ASCET (classes, modules, projects, etc.) in a database. The Component Manager allows you to manage the database items from a comprehensible user interface. Similar to the Windows Explorer, you can create folders and subfolders, move, copy, import, and export individual items, and also create entirely new databases. This means that you can organize your data in a similar fashion as you are accustomed to for the file system.

Besides organizing your data, you can also edit components and projects in ASCET.

Database Items

An ASCET database contains different types of database items. This section presents an overview of the types of database items available in ASCET.

Components: The specification of an embedded software system in ASCET is made up of components. A component is a modular piece of functionality which contains algorithms and data. The different algorithms specified within a component can be executed independently of each other. The components of an Embedded Control System can be combined into projects.

On a physical level, components are specified either graphically as block diagrams or state machines, or in ESDL-Code. Alternatively, they can be specified in C code. Components that are specified either graphically or in ESDL are implementation-independent, i.e. they can be used to generate code for different platforms. Components specified in C code are always platform-dependent. They encapsulate target-specific behavior.

The types of component available in ASCET and their usage are described in chapter "Components" in the ASCET reference manual. Information on working with components is available in the description of the relevant editor.

Projects: A project specifies the functionality of an Embedded Control System. It contains all the components together with the necessary protocols, operating system and code generation settings. A project determines the communication between components and the order in which algorithms are executed.

The project concept is explained in chapter "Projects" in the ASCET reference manual; a description of how to work with projects can be found in chapter "The Project Editor" on page 371.

Icons: When a component is nested in another component, it appears as a block in the diagram of the enclosing component. You can assign an icon to a nested component to make a complex diagram more readable.

A selection of icons is provided with ASCET. In addition, users can define and edit their own. Importing, creating and editing icons is described in section "The Icon Editor" on page 558.

Signals: In order to test ASCET models under realistic conditions, genuine measurement data can be imported into ASCET, and used as input in the testing of ASCET models. This data is stored in signal items.

Additional information on working with signals can be found in section "The Signal Viewer" on page 555.

Container: Containers are used 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. Details are given in chapter "Containers" on page 443.

Enumerations: Enumerations are unique types with values taken from a group of known constants called enumerators.

ASAM-MCD-2MC project: The ASAM-MCD-2MC file represents the interface between ASCET-MD and other programs (INCA, ASCET-RP, and others) that recognize the standard ASAM-MCD format.

2.3.1 Managing Database Items

In ASCET, database items are organized in folders. A database can contain any number of top-level folders, which in turn can contain other folders. Database items must be stored in folders, they cannot be created at the root level of a database. Database items are identified by their object IDs. Even though identical names are not allowed in the same folder, objects in different folders can have identical names.

This section explains the standard operations you can perform on database items.

To create a folder:

- In the Component Manager, "1 Database" list, select the database name or the folder you want the folder to be in.

  You can place folders only beneath the database name.

- Select **Insert** → **Folder**

or

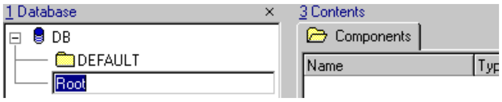
- click on the **Insert Folder** button

or

- press the <INSERT> key.

  A new folder appears in the "1 Database" list. When you place the folder at the top hierarchy level (see figure), `Root` is used as default name, otherwise, the default name is `Folder`.

- Edit the folder name. You can simply type over the highlighted name and then press <ENTER>.



Note

Folder and component names are not case-sensitive. Folder names differing only in the use of upper and lower case letters are not allowed.

Each database has to have at least one top-level folder that is created automatically when you create the database.

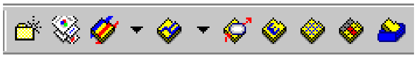
To create a database item:

- In the "1 Database" list, select the folder you want the new item to be in.

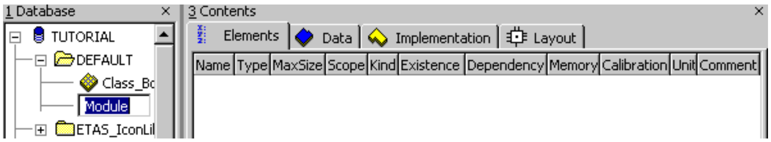
- Select **Insert** → ***<component type>*** or **Insert** → ***<component type>*** → ***<item type>***

or

- click on the corresponding button in the Insert button bar



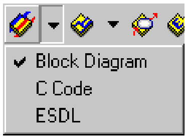
The new item is created with a default name and layout.



- Edit the name and press <ENTER>.

If you create a component of type `Module`, `Class` or `Continuous Time Block`, you can choose whether the item should be realized as a block diagram, in ESDL or C code. These item types are available via the ***<item type>*** submenus or the **Insert Module -** ***<item type>*** and **Insert Class -** ***<item type>*** buttons. The default item type of the buttons can be adjusted.

To select the default item type of classes/modules:



- Click on the arrow button next to **Insert Module -** ***<item type>*** or **Insert Class -** ***<item type>***.

  A drop-down list opens. The current default is checked.

- Select the default type for modules or classes.

  When you now use the buttons to create components, these have the default item type.-

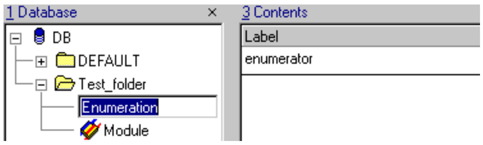
To create an enumeration:

- In the "1 Database" list, select the folder you want the enumeration to be in.
- Select **Insert** → **Enumeration**

or

- click on the **Insert Enumeration** button.

  The enumeration is created with a default name; the first enumerator is displayed in the "3 Contents" field.



- Click anywhere in the "3 Contents" field to focus on it.

  The menu changes described on page 28 become effective.

- Select **Component** → **Enumeration** → **Add***

or

- press <INSERT>

or

- right-click in the "3 Contents" field and select **Add** → ***<submenu>*** from the context menu to add an enumerator.

  The functions in **Add** → ***<submenu>*** place the new enumerator at the following positions:

as first	in the first row
as last	in the last row
before selection	before the selected enumerator
after selection	after the selected enumerator

- Select **Edit** → **Rename**

or

- press <F2> to rename a selected enumerator.
- Select **Edit** → **Delete**

or

- press <DEL> to delete a selected enumerator.

- Select **Component** → **Enumeration** → **Shift Up** to move an enumerator up the list.
- Select **Component** → **Enumeration** → **Shift Down** to move an enumerator down the list.

  All menu items are also available as context menu in the "3 Contents" field.

To edit a database item:

- In the "1 Database" list, select the entry you want to edit.
- double-click on the entry

or

- select **Component** → **Edit Item**

or

- press the <RETURN> key.

  The editor for the selected item opens.

To rename a folder or database item:

- In the "1 Database" list, select the folder or item you want to rename.
- Select **Edit** → **Rename**

or

- Press <F2>.

  The name of the folder is highlighted.
- Edit the name and press <ENTER>.

To delete a folder or database item:

Note

Folders and components are deleted directly from the database and cannot be recovered. When you delete a folder, all items in that folder are deleted as well. Therefore, use the "Delete" command with care.

- In the "1 Database" list, select the folder or item you want to delete.
- Select **Edit** → **Delete**

or

- click on the **Delete** button

or

- press the <DEL> key.

  A confirmation window is displayed when a component or a folder containing folders or items is to be deleted.
  Empty folders are removed without confirmation.

- Click **OK** to confirm the deletion.

You can copy and move database items or entire folders and their contents. When the target directory contains an object with the same name as the original, the extension `"_1"` is added to the name of the copied object.

Note

*Top-level folder can be copied and shifted **only** at top-level folder level with **Cut**/**Copy**/**Paste**.*
*Subfolders **cannot** be lifted to top-level folder level via **Cut**/**Copy**/**Paste**.*

To copy a database item or folder:

- Select the item or folder you want to copy.
- Select **Edit** → **Copy**

or

- click on the **Copy** button

or

- press <CTRL> + <C>.

  The database item or folder is copied to the clipboard.

To cut a database item or folder:

- Select the item or folder you want to copy.
- Select **Edit** → **Cut**

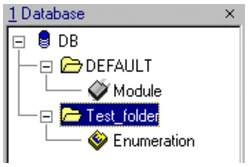
or

- click on the **Cut** button

or

- press <CTRL> + <X>.

  The database item or folder is moved to the clipboard. The item symbol in the Component Manager appears now black and white.



Note

*When you close the database while the item or folder is still in the clipboard, the clipboard is emptied, but the object is **not** deleted.*

To insert a database item or folder:

- In the Component Manager, select the target folder for the database item or folder in the clibboard.
- Select **Edit** → **Paste**

or

- click on the **Paste** button

or

- press <CTRL> + <V>.

  The database entry or folder is inserted in the selected target folder. When the item was cut (cf. page 79), the corresponding entry in the target folder is deleted.

When items or folders are moved within the same database, item and/or folder names are usually retained in the target folder. The new item is automatically renamed only if the target folder already contains an item or a folder with the same name, thus avoiding naming conflicts.

If you want to re-implement an existing block diagram component in C code or ESDL code (or vice-versa), you can copy the structure of the original component, so you will not have to specify it again. When you copy the structure of a component, the entire interface of the component is created automatically in the target component.

To copy the structure of a component:

- In the Component Manager, select the component you want to copy.
- Select **Component** → **Reproduce As** → ***<item type>*** to determine the item type to be created.

  The new component is created in the same folder as the original one. It is named as the original, with the extension "`_1`".

  The new component contains the same interface as the original one, but no functionality is specified for the component.
- You may want to rename the new component to avoid naming conflicts.
- Double-click on the new item to open the relevant component editor and specify the functionality you want.

To save the current database:

When you make changes to ASCET databases, those changes are stored in a cache, in RAM. To make the changes permanent, you have to save them to the database on the hard disk.

- Select **File** → **Save Database**

or

- click on the **Save** button

or

- press <CTRL> + <S>.

  The cache content is written to the hard disk.

For information on saving your database automatically at regular intervals, see section "General Options" on page 41.

2.3.2 Working with View Concepts

You can work with view concepts via menu functions or via context menus. A description of the menu functions is given in section "Description of Menu Options" on page 19.

Editing Components and Projects in the Component Manager

In ASCET it is possible to modify the properties of components and projects in the Component Manager. By selecting the element you want in the "3 Contents" field, you can edit the configuration, the data set, the implementation or the layout, depending on the view you selected.

Method arguments and return values are exceptions to this. These elements can only be changed in the specification editors.

To edit database items:

- Highlight a folder in the "1 Database" field.

  The folder view is shown in the "3 Contents" field.
- In the "Components" tab, highlight an entry.
- Select **Component** → **Edit Item**

or

- press <RETURN>

or

- double-click on the highlighted entry.

  The editor for the selected item opens.
- Select **Rename** from the context menu

or

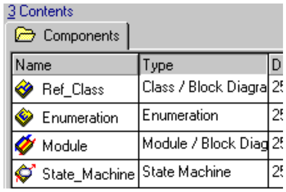
- press <F2> to rename the selected item.
- From the context menu, select **Select All** to select all entries.
- Select **Delete** from the context menu

or

- press <DEL> to delete the selected items.
- Select **Sort by** → ***<column>*** from the context menu

or

- click on a column name to sort the display by that column.



3 Contents

Components

Name	Type	D
Ref_Class	Class / Block Diagra	25
Enumeration	Enumeration	25
Module	Module / Block Diag	25
State_Machine	State Machine	25

In an ascending sorting, numbers come before upper-case letters, which come before lower-case letters.

A second sorting reverts the sorting order.

To edit elements:

- Select a component or project in the "1 Database" field.
- In the "3 Contents" field, click on the "Elements" tab.

  The element view is displayed.
- Click on a column name to sort the display by that column.
- In the "Elements" tab, select an element.
- Select **Component** → **Edit Item**

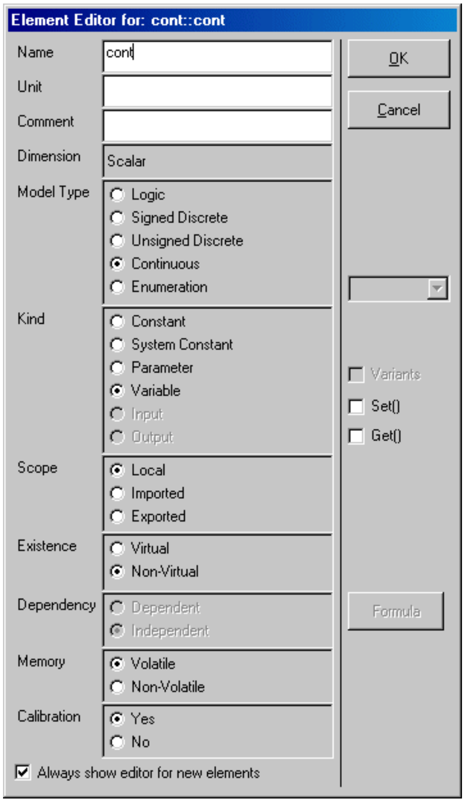
or

- press <ENTER>

or

- double-click on the selected element.

  The element editor opens.


Element Editor for: cont::cont

Name: cont
Unit:
Comment:
Dimension: Scalar
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
☐ Variants ☐ Set() ☐ Get()
[✓ Always show editor for new elements]

Chapter 4.10 contains detailed information about editing element configurations.

- Select **Edit** → **Rename**

or

- press <F2> to rename the selected element.
- From the context menu, select **Select All** to select all entries.
- Select **Edit** → **Delete**

or

- press <DEL> to delete the selected elements.

- Select **Edit** → **Copy**

or

- press <CTRL> + <C> to copy the selected elements to the clipboard.
- Select **Edit** → **Paste**

or

- press <CTRL> + <V> to paste elements from the clipboard to the component.

  If an element with the same name already exists, a counter (_n) is added to the name of the pasted element.

To edit the data:

- In the "1 Database" field, select a component or project.
- In the "3 Contents" field, click on the "Data" tab.

  The data view is displayed.
- Click on a column name to sort the display by that column.
- Select the element you want.
- Select **Component** → **Edit Item**

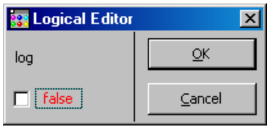
or

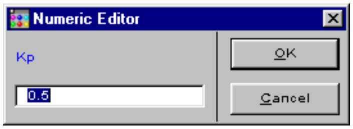
- press <ENTER>

or

- double-click on the selected element.

  The appropriate data editor opens, depending on the kind of the selected element.
  For example, the "Logical Editor" or "Numeric Editor" dialog opens if you have selected a logical or scalar element.





Chapter 4.11 contains detailed information about editing data and data sets.

- Select **Edit** → **Rename**

or

- press <F2> to rename the selected element.
- Select **Edit** → **Copy**

or

- press <Ctrl> + <c> to copy the current data of the selected element to the clipboard.
- From the context menu, select **Select All** to select all entries.
- Select **Edit** → **Paste**

or

- press <Ctrl> + <v> to copy the data from the clipboard to the selected elements.
- Select **Edit** → **Delete**

or

- press <Del> to delete the selected elements.

To edit the implementation:

- In the "1 Database" field, select a component or project.
- In the "3 Contents" field, click on the "Implementation" tab (implementation view).
- Click on a column name to sort the display by that column.
- Select the element you want.

  You can select both basic elements and included components.
- Select **Component** → **Edit Item**

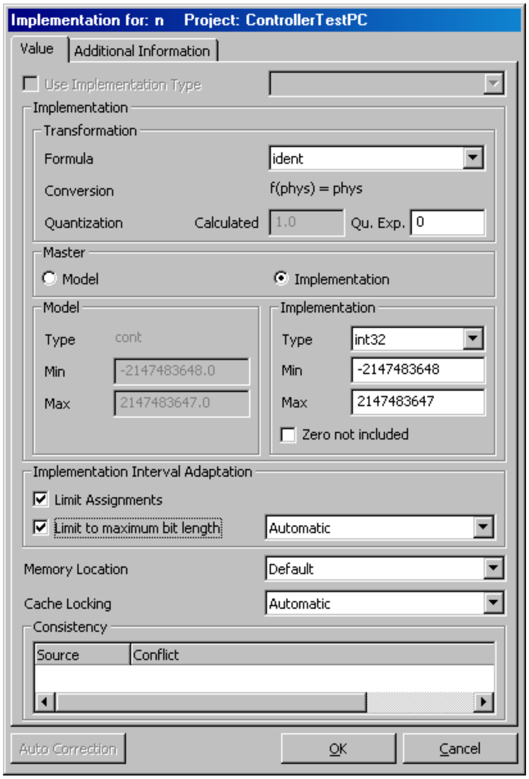
or

- press <ENTER>

or

- double-click on the selected element.

  The corresponding implementation editor opens (the figure shows the implementation editor for basic elements).



Implementation for: n   Project: ControllerTestPC

Value | Additional Information

☐ Use Implementation Type

Implementation

Transformation
- Formula: ident
- Conversion: f(phys) = phys
- Quantization: Calculated 1.0, Qu. Exp. 0

Master: ○ Model ● Implementation

Model		Implementation	
Type	cont	Type	int32
Min	-2147483648.0	Min	-2147483648
Max	2147483647.0	Max	2147483647
		☐ Zero not included	

Implementation Interval Adaptation
- ☑ Limit Assignments
- ☑ Limit to maximum bit length: Automatic

Memory Location: Default

Cache Locking: Automatic

Consistency

Source	Conflict

Chapter 4.12 contains detailed information about editing implementations.

- Select **Edit** → **Rename**

or

- press <F2> to rename the selected element.
- Select **Edit** → **Copy**

or

- press <CTRL> + <C> to copy the current implementation of the selected element to the clipboard.

- From the context menu, select **Select All** to select all entries.
- Select **Edit** → **Paste**

or

- press <Ctrl> + <v> to copy the implementation from the clipboard to the selected elements.

Note

You can copy and paste implementations only between elements of the same type. It is ***not*** *possible to copy the implementation of one element type (e.g., a characteristic field) to an element of a different type (e.g., scalar, matrix, ...).*

- Select **Edit** → **Delete**

or

- press <Del> to delete the selected elements.

To edit the layout:

- In the "1 Database" field, select a component.

Note

The "Layout" tab does not exist for projects.

- In the "3 Contents" field, click on the "Layout" tab.

  The layout view is displayed.
- Select **Component** → **Edit Layout**

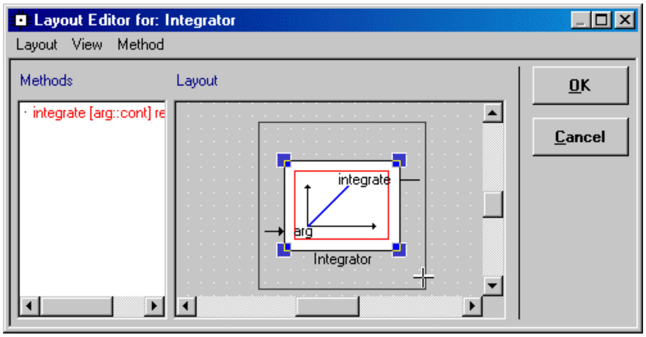
or

- press <ENTER>

or

- double-click on the layout displayed.

  The layout editor opens.



Chapter 4.13 contains detailed information about editing component layouts.

Selecting Another Data or Implementation Set

If more data or implementation sets are created for a component, you can select these using a combo box. This combo box is only available in the data and implementation views.

To select another data or implementation set:

- In the "Data" or "Implementation" tab, click on the combo box.
- Select a data or implementation set.

  The values in the individual elements are changed accordingly.

2.3.3 Exporting Folders and Database Items

In ASCET, it is possible to exchange folders and database items between databases. You can export entire folders, individual database items, or any selection of database entries and folders. A distinction is made between exporting individual items (*flat* export), or the items together with all other items referenced by them (*recursive* export).

Note

During export, the content of the "2 Comment" field (cf. page 16) is exported only for top-level folders and items. Comments on subfolders are not exported.

Binary Export

The `*.exp` export file format is a very space-efficient binary format that can be generated and read quickly. However, generating such export files can be a memory-intensive task. If you are exporting large folders, you can distribute their contents over several files to speed up the export/import process. You can distribute the contents of large folders over several files by automatically generating one file per exported item.

When the **Write XML file on Binary Export** or **Write ASCII file on Binary Export** option has been activated in the "Options" window (see "Export Options" on page 61), description files in the selected format are created in addition to the export file. These description files can be viewed with an internet browser or a text editor, respectively.

Note

The generation of XML description files requires at present Microsoft Internet Explorer V5.5 (with MSXML Parser V3.0) or V6.

AMD Export

With ASCET 5.2, the XML-based export format `*.amd` is provided. When you select this export format, several `*.amd` files are created for each exported component. These files store the complete model description. The names of these files are created as follows:

```
<component name>.<information type>.amd
```

Tab. 2-3 lists the information types and file content.

Information type	Content
`main`	names and properties of the elements in the project/component (cf. page 447)
`data`	data of the elements in the component/project (cf. page 458)
`experiments`	experiment environments defined in the component/project (cf. page 561)
`implementation`	implementations of the elements in the component/project (cf. page 475)
`specification`	specification details (e.g., interface information, block diagram structure, code of an ESDL or C code component)

Information type	Content
`project`[a]	project-specific data such as operating system configuration (cf. page 390), project settings (cf. page 407)
`project.formulas`[a]	formulas defined in the project (cf. page 424)
`project.implementationTypes`[a]	implementation types defined in the project (cf. page 430)

a. for projects only

Tab. 2-3 Types of AMD export files

Each AMD file contains a signature which is used, during import, to check whether the file was changed between export and import.

If desired, the AMD files can be collected and compressed into a Zip file during export. This Zip file has the extension `*.axl`.

Other Export Formats

Besides the export formats mentioned above, an XML format (`*.xml`) is available, too, that corresponds to the optional XML description file for binary export. This XML format is fundamentally different from the AMD format; it cannot be re-imported into ASCET.

For the export of ASAM-MCD-2MC projects, the export format `*.a2l` is available.

Note

If you use the `*.a2l` format to export something other than an ASAM-MCD-2MC project, you prodce an error.

Performing the Export

Before you export folders or database items, select the appropriate export mode (with or without referenced items), and the file distribution, from the export options (see "Export Options" on page 61)

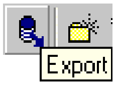
To export a folder or database item:

- In the "1 Database" list of the Component Manager, select the folders or items to be exported.
- Select **File** → **Export**

or

- select **Export** from the context menu

or

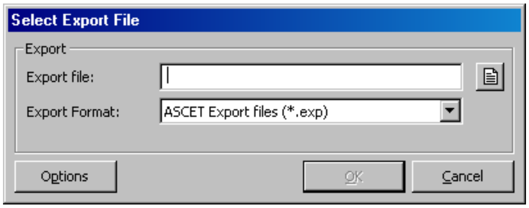


- click on the **Export** button

or

- press <CTRL> + <E>.

  The "Select Export File" window opens.



Export file:

Export Format: ASCET Export files (*.exp)

- In the "Export Format" combo box, select an export format.

`ASAP2 files (*.a2l)`	only ASAM-MCD-2MC projects
`ASCET Model Data files (*.amd)`	AMD export (cf. page 90)
`ASCET compressed Model Data files (*.axl)`	compressed AMD export
`ASCET Export files (*.exp)`	binary export (cf. page 90)
`ASCET XML files old format (*.xml)`	XML export, *can-not* be imported

- In the "Export File" field, enter path and filename of the export file manually or via the 🗎 button.

Note

When the export option **One File for each Item** *is activated, the field is named "Export folder", andthe button looks like this:* 📁

The **OK** button is now available.

- Click **Options** if you want to adjust the export options.

  The "Options" window opens in the "Export" node (see section "Export Options" on page 61).

  – Set the export options.

  – Click **OK** to close the "Options" window.

- In the "Select Export File" window, click **OK** to start the export.

  The objects are exported to the file and folder you selected.
  During AMD exort, a series of `*.amd` files is created for each exported object.

Note

When exporting database items on a "one file per item" basis, or when using AMD export including referenced items, you should export to an empty directory to keep your exported files manageable. ASCET automatically exports to the directory specified in "Default Export Path".

2.3.4 Importing Folders and Database Items

Every object in an ASCET database has a unique identity tag, and is known to the database only by this identity tag. Database items reference each other only on the basis of these tags, not on the basis of the names displayed in the Component Manager.

You can import from one or more export files of the following formats into your database.

binary export file (s. page 90)	`ASCET Export files (*.exp;*.prj)`
AMD export file (s. page 90)	`ASCET Model Data files (*.main.amd; *.main.xml)`
compressed AMD export file	`ASCET compressed Model Data files (*.axl;*.zip)`
ASAM-MCD-2MC project	`ASAP2 files (*.a2l)`

The XML export format (s. page 91) *cannot* be imported.

The folders and items required are created automatically. You are prompted for confirmation when an existing item is overwritten by an imported item.

Besides the components themselves, the export files also store the component paths in the exporting database. If an imported item is located in a folder existing in the target database, it is written into that folder. If an item is located in a folder in the source database that does not exist in the target database, the folder is automatically created in the target database. If necessary, several levels of folders are created. The folder hierarchy is recreated as it exists in the source database.

When an item is imported that already exists in the target database, the existing item is overwritten. Renaming offers no protection against overwriting because database items are identified by their object IDs, not their names.
With AMD import, the import option **Remap OID** makes sure that an existing object is not overwritten, but a copy with a new object ID is created.

If an imported item overwrites an existing item, the target database path and the behavior of existing implementations can be specified in the import options (see "Import Options" on page 63). When you deactivate the **Keep Folder Path in Database** option, the imported item is stored at the export database path, even in case it is stored in a different folder in the target database. To keep the existing path name, **Keep Folder Path in Database** has to be activated.

It is possible to protect data in a database from being overwritten in this way.

To disallow overwriting of data in a database:

- In the Component Manager, select the items and folders you want to protect.
- Select **Component** → **Disallow Import**.

  The items and folders selected are protected from being overwritten on import. Protected items are marked with `<Disallow Import>` in the "1 Database" list.

If a folder has been set to disallow import, this simply means that the folder itself cannot be overwritten by an imported folder.

Performing the Import

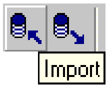
Before you import folders or items, set the import options in the user options (see "Import Options" on page 63).

To import from export files:

- In the Component Manager, open the target database for the import.

- For AMD import or import of ASAM-MCD-2MC files, select a folder.
- Select **File** → **Import**

or

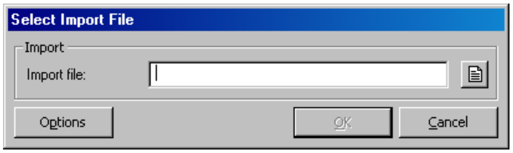


- click on the **Import** button

or

- press <CTRL> + <M>.

  The "Select Import File" window opens.



- In the "Import File" field, enter path and file-name of the file you want to import manually or via the button.

Note

You can enter or select more than one file.

The file format does not have to be explicitly specified. It is identified by the file extension.

The **OK** button is now available.

- Click **Options** if you want to adjust the import options.

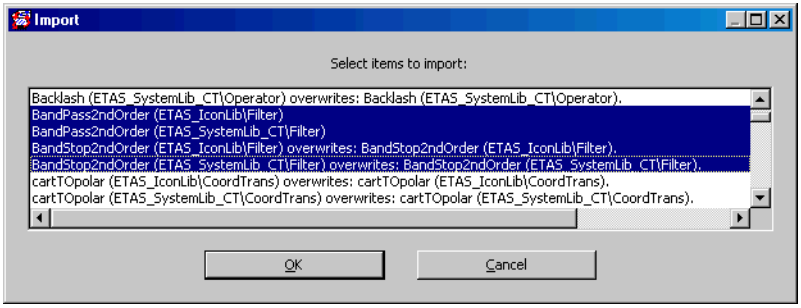
  The "Options" window opens in the "Import" node (see section "Import Options" on page 63).

  – Set the import options.

  – Click **OK** to close the "Options" window.

- In the "Select Import File" window, click **OK** to start the import.

  The "Import" or "Items available for import" window lists the content of the file(s). If one of the objects already exists in the database (identical object ID), the window shows the item to be overwritten.



Import

Select items to import:

```
Backlash (ETAS_SystemLib_CT\Operator) overwrites: Backlash (ETAS_SystemLib_CT\Operator).
BandPass2ndOrder (ETAS_IconLib\Filter)
BandPass2ndOrder (ETAS_SystemLib_CT\Filter)
BandStop2ndOrder (ETAS_IconLib\Filter) overwrites: BandStop2ndOrder (ETAS_IconLib\Filter).
BandStop2ndOrder (ETAS_SystemLib_CT\Filter) overwrites: BandStop2ndOrder (ETAS_SystemLib_CT\Filter).
cartTOpolar (ETAS_IconLib\CoordTrans) overwrites: cartTOpolar (ETAS_IconLib\CoordTrans).
cartTOpolar (ETAS_SystemLib_CT\CoordTrans) overwrites: cartTOpolar (ETAS_SystemLib_CT\CoordTrans).
```

- Select all the items to be imported and click **OK**.

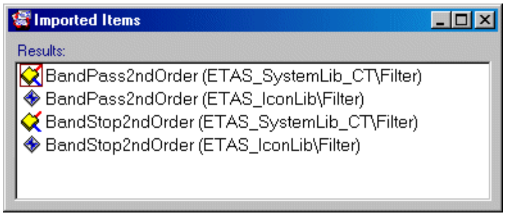
  If you are importing existing objects, you are asked to confirm the overwriting of the existing database items.

Note

When the import option **Remap OID** *is selected for AMD import, an existing object is not overwritten, but a copy wih new object ID is created.*

- Confirm the message with **OK**.

  The items selected in the "Import" window are now imported. Finally, the imported items are listed in the "Imported Items" window.

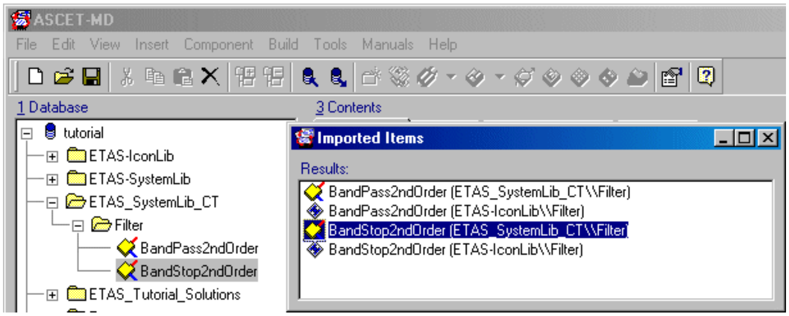


Imported Items

Results:

```
BandPass2ndOrder (ETAS_SystemLib_CT\Filter)
BandPass2ndOrder (ETAS_IconLib\Filter)
BandStop2ndOrder (ETAS_SystemLib_CT\Filter)
BandStop2ndOrder (ETAS_IconLib\Filter)
```

- When you click on an item in the "Imported Items" window, it is highlighted in the Component Manager.



BandPass2ndOrder (ETAS_SystemLib_CT\\Filter)
BandPass2ndOrder (ETAS-IconLib\\Filter)
BandStop2ndOrder (ETAS_SystemLib_CT\\Filter)
BandStop2ndOrder (ETAS-IconLib\\Filter)

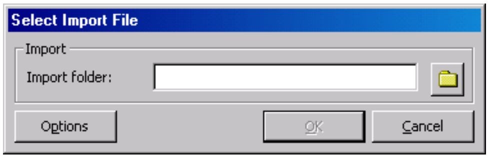
Importing a Directory Content

To allow an easy import of all export files in a given directory, the menu option **File** → **Import directory.** is provided

Importing a directory content:

- In the Component Manager, select **File** → **Import directory**.

  The "Select Import File" window opens.



- In the "Import folder" field, enter path and filename of the file you want to import manually or via the button.

  The **OK** button is now available.

- Click **Options** if you want to adjust the import options.

  The "Options" window opens in the "Import" node (see section "Import Options" on page 63).

  – Set the import options.

- Click **OK** to close the "Options" window.

For the first import file, an "Import" or "Items available for import" window opens that lists the content of the file. If one of the objects already exists in the database (identical object ID), the window shows the item to be over-written.

- Select all items to be imported and click **OK**.

  If you are importing existing objects, you are asked to confirm the overwriting of the exist-ing database items.

- Confirm the message with **OK**.

  The selected items are now imported. After that, an "Import" or "Items available for import" window opens for the next import file.

  After all imports, the imported items are listed in the "Imported Items" window.

Special Features of the AMD Import

During import from AMD files, several tests are performed.

Integrity check: The signatures are used to check whether the files have been changed by third parties.

Consistency check: The system checks for the following errors:

- invalid file

  If one of the AMD files is invalid, i.e. unreadable, not XML conforming, or not conforming to the respective XML schema, the following hap-pens.

  – The component is not imported.

  – The"Import Problems" window lists file and problem.

  – No automatic solution exists.

- missing file

  An ASCET component is specified by several files (see "AMD Export"). If one or more of these files are missing, the following happens.

  – The component is not imported.

  – The"Import Problems" window lists the missing files.

  - The user is offered the opportunity to create default files for the missing files.
- missing components during import

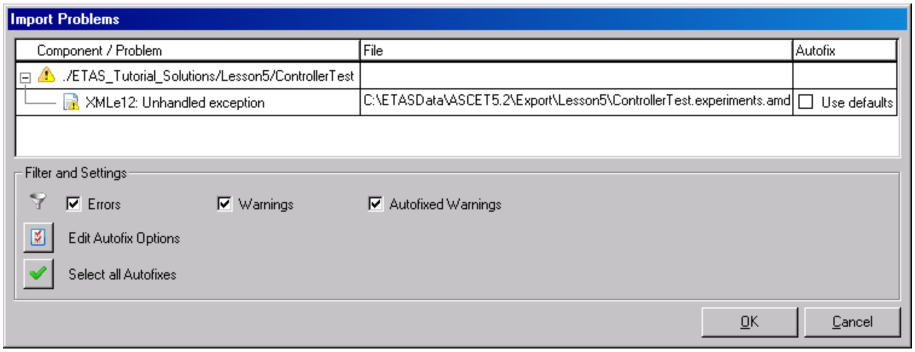
  If a referenced component does not exist, or is unavailable, the "Import Problems" window lists the missing files.
- missing information in a file

  If information is missing in an AMD file (e.g., an element was defined, but its data are missing), but the file itself is valid (i.e. readable, conforming to XML and the respective schema), the following happens.
  - The component is not imported.
  - The"Import Problems" window lists the missing information.
  - The user is offered the opportunity to create default values for the missing information.
- wrong information in a file

  If an AMD file contains wrong information (e.g., data that do not agree with infoation in other AMD files of the component), the following happens.
  - The component is not imported.
  - The"Import Problems" window lists the wrong information.
  - No automatic solution exists.
- obsolete information in a file

  If an AMD file contains more information than required (e.g., a data configuration for a non-existing element), the following happens.
  - The component is not imported.
  - The"Import Problems" window lists the obsolete information.
  - The user is offered the opportunity to ignore the obsolete information.

The "Import Problems" window contains the following elements.



Import Problems

Component / Problem	File	Autofix
⚠ ./ETAS_Tutorial_Solutions/Lesson5/ControllerTest		
⚠ XMLe12: Unhandled exception	C:\ETASData\ASCET5.2\Export\Lesson5\ControllerTest.experiments.amd	☐ Use defaults

Filter and Settings

☑ Errors ☑ Warnings ☑ Autofixed Warnings

Edit Autofix Options

Select all Autofixes

OK Cancel

- "Component / Problem" column

  Affected component and kind of problem, e.g., `unhandled exception`, `Invalid AMD file (missing)`, `Data for Element <name> missing` or `Obsolete data for undefined element <name>`.

  Problems can be of type warning (symbole ⚠ and ⚠) or error (symbols ✖ and ✖).

- "File" column

  Path and name of the affected file.

- "Autofix" column

  Automatic solution; the possible solutions must be activated for the actions to be performed.

  The column is empty for problems without automatic solution.

- context menu in the table

  – **Expand all Nodes** (<ALT>+<*>)

    Expands all nodes in the "Component / Problem" column.

  – **Select All Autofixes** (<CTRL>+<A>)

    Activiates all automatic solutions in the "Autofix" column.

  – **Select Autofixes of this Type** (<ALT>+<A>)

    Only available when an entry in the "Autofix" column is selected. Activates all automatic solutions of the selected type.

- **Always Autofix Problems of this Type**

  Only available when an entry in the "Autofix" column is selected. Ensures that the solutions of the selected type are, from now on, performed automatcally; you do not have to activate these options in the "Import Problems" window.

- **Save Issue List as XML** (<CTRL>+<S>)

  Saves the window content to an XML file. For some errors, you get additional information.

  Such an XML file is helpful when you intend to analyze the problems automatically.

- Filter options
  - **Errors**

    Shows problems of type error.

  - **Warnings**

    Shows problems of type warning.

  - **Autofixed Warnings**

    Shows automatically solved problems of type warning.

- **Edit Autofix Options** button (☑)

  Opens the "Options" window where settings for automatic problem solutions can be made.

- **Select all Autofixes** button (✓)

  Activates all automatic solutions in the "Autofix" column.

- **OK** button

  Closes the window and performs the activated automatic solutions.

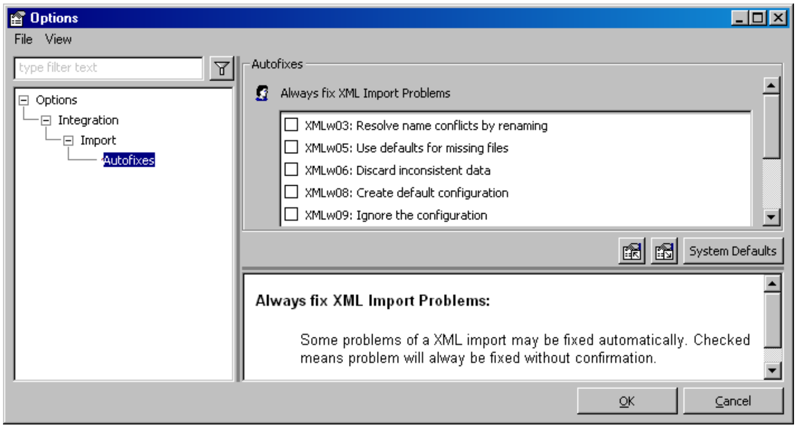
- **Cancel** button

  Closes the window without performing the activated automatic solutions.

To set options for automatic problem solutions:

- In the "Import Problems" window, click on **Options**.

  The "Options" window opens. It contains only the "Import" node.



Autofixes

Always fix XML Import Problems

- [ ] XMLw03: Resolve name conflicts by renaming
- [ ] XMLw05: Use defaults for missing files
- [ ] XMLw06: Discard inconsistent data
- [ ] XMLw08: Create default configuration
- [ ] XMLw09: Ignore the configuration

Always fix XML Import Problems:

Some problems of a XML import may be fixed automatically. Checked means problem will alway be fixed without confirmation.

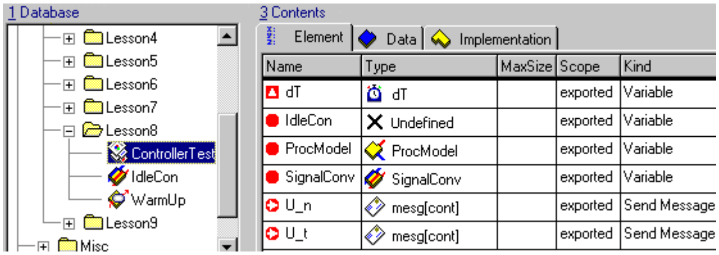
The "Always fix AMD Import problems" list contains solutions for the problems possible during AMD import.

- Activate the options for all problems that shall be solved without being displayed in the "Import Problems" window.
- Close the "Options" window with **OK**.

  During the next AMD import, the selected solutions are performed automatically; you do not have to activate options in the "Import Problems" window.

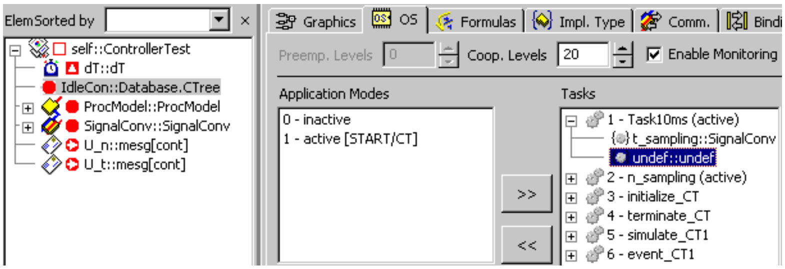
Importing Projects

Projects are imported like all other database entries. Any modules belonging to a project which are not available in the database after an import are shown in the element display of the Component Manager.



Name	Type	MaxSize	Scope	Kind
dT	dT		exported	Variable
IdleCon	Undefined		exported	Variable
ProcModel	ProcModel		exported	Variable
SignalConv	SignalConv		exported	Variable
U_n	mesg[cont]		exported	Send Message
U_t	mesg[cont]		exported	Send Message

If the project is opened in the project editor (chapter 4.8 "The Project Editor" on page 371), the names of the missing modules are in the "Elements" list. In the operating system editor ("OS" tab, s. "Defining the Scheduling in the OS Editor" on page 390) the missing processes are deleted from the "Processes" field; they are however retained in the task list as open references (`undef::undef`).



Preemp. Levels 0 Coop. Levels 20 Enable Monitoring ☑

Application Modes
0 - inactive
1 - active [START/CT]

Tasks
1 - Task10ms (active)
t_sampling::SignalConv
undef::undef
2 - n_sampling (active)
3 - initialize_CT
4 - terminate_CT
5 - simulate_CT1
6 - event_CT1

You can import the missing modules at a later time even if you have opened the project. Afterwards, the project has no open references, and the processes are automatically inserted into the right tasks.

Importing Items from Old ASCET Versions

Export files created with versions prior to ASCET-SD 4.0 cannot be imported into ASCET 5.2 because the current version offers no interface to the old database formats. When you try to import such a file as described on page 94, you get the following error message:

```
This file is compatible with ASCET-SD V3.0 or earlier.
It cannot be imported directly. Convert it to a data-
base using your old version of ASCET-SD, then open
this database.
```

To import an export file from ASCET-SD versions prior to 4.0:

If you still have ASCET-SD 4.x, you can use very old export files with ASCET 5.2. Proceed as follows:

- Start ASCET-SD 4.x.

  These versions contain interfaces to the very old database formats.
- Import the old export file.

  The database items are converted to the format of the corresponding ASCET-SD version.
- Export the imported items.
- Close ASCET-SD 4.x.
- Start ASCET 5.2.
- Import the database items you have just exported from ASCET-SD 4.x, as described on page 94.

2.3.5 Working with Database Items

References on Items

A reference to an item is created whenever the item is used by or included in another item (e.g. a module that is included in a project is referenced by that project). When moving items between folders, or renaming existing items in folders, all references to that item are updated automatically.

The Component Manager does not automatically resynchronize with the database if references have been modified. To ensure the consistency of references in the Component Manager, you need to update its references explicitly.

To update references in the Component Manager:

- In the Component Manager, select **View** → **Update**

or

- Press <F5>.

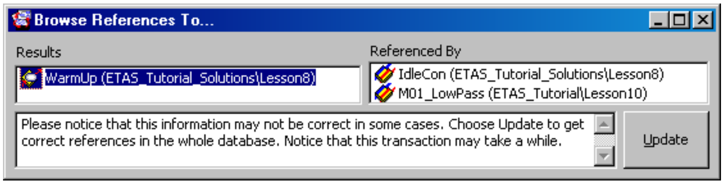
  The Component Manager display is compared with the database, and updated if necessary.

You can view a list of references to ensure that your system is consistent before removing items or folders from the database.

To display the references to a database item:

- Select a database item.
- Select **Component** → **Show References**.

  The "Browse References To" dialog opens.



WarmUp (ETAS_Tutorial_Solutions\Lesson8)

IdleCon (ETAS_Tutorial_Solutions\Lesson8)
M01_LowPass (ETAS_Tutorial\Lesson10)

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.

It shows the names of all the items which reference the selected item.

Since references can be cyclic; an item can reference an item that references it. When you delete an item, you should always make sure that the referenced components are not corrupted. ASCET displays a warning and a list of references if you attempt to remove a component that is referenced by other components.

Deleting database items that are referenced by other components can destroy the referencing components. This can be prevented by replacing the component you want to remove with a new one or a different one.

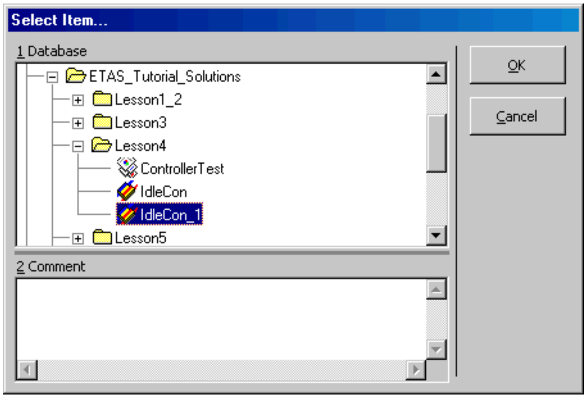
When a component is replaced, all the database items that have references to the replaced component adjust their references to the replacing component. The replaced component is no longer referenced by any other database item.

To replace the references to a database item:

- In the "1 Database" list of the Component Manager, select the item you want to replace.

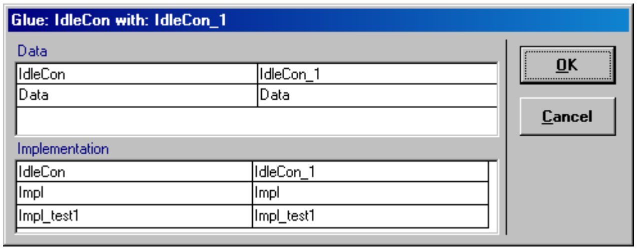
- Select **Component** → **Replace References**.

  The "Select Item" window opens.



- Select the item intended to replace to the first item and click **OK**.
- In the "Confirm" window, confirm the command with **OK**.

  The "Glue: *<item>* with: *<item>*" window opens. All replacements are listed here.


Glue: IdleCon with: IdleCon_1

Data

IdleCon	IdleCon_1
Data	Data

Implementation

IdleCon	IdleCon_1
Impl	Impl
Impl_test1	Impl_test1

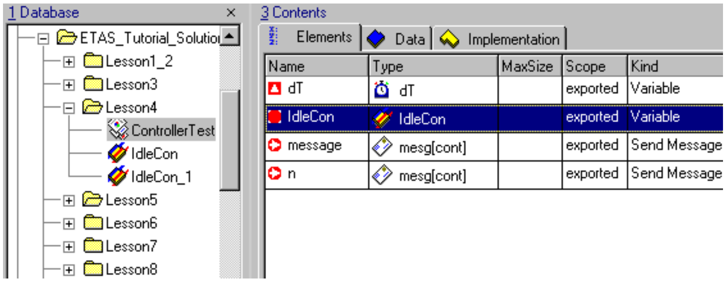
- Confirm by clicking **OK**.

  The references to the first item are replaced throughout the database.

With this procedure, you only replace the reference to one item by a reference to another item. Replacing references does not affect the database items in the sense that any of the items involved no longer exists or has moved to another location.

Here is an example to make the way **Component → Replace References** works clearer.

The illustration shows a section from the element view of the Component Manager for the `ControllerTest` project in the `Lesson4` folder of the tutorial database. This project references the `IdleCon` component.


1 Database

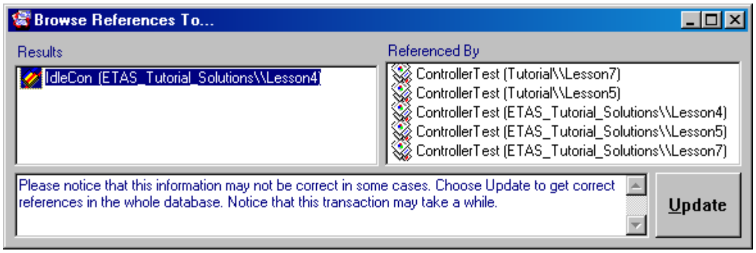
- ETAS_Tutorial_Solutio
  - Lesson1_2
  - Lesson3
  - Lesson4
    - ControllerTest
    - IdleCon
    - IdleCon_1
  - Lesson5
  - Lesson6
  - Lesson7
  - Lesson8

3 Contents

Elements | Data | Implementation

Name	Type	MaxSize	Scope	Kind
dT	dT		exported	Variable
IdleCon	IdleCon		exported	Variable
message	mesg[cont]		exported	Send Message
n	mesg[cont]		exported	Send Message

After the selection of `IdleCon` in the "1 Database" list, this can be checked with **Component → Show References**.


Browse References To...

Results

IdleCon (ETAS_Tutorial_Solutions\\Lesson4)

Referenced By

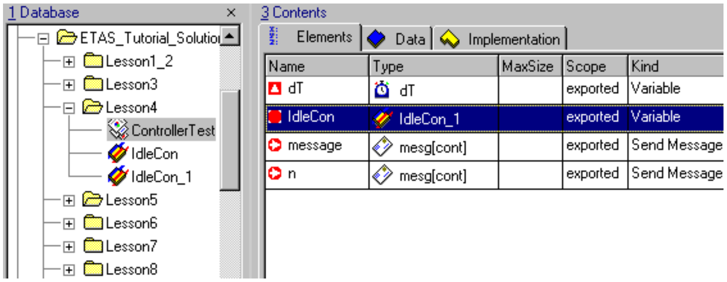
ControllerTest (Tutorial\\Lesson7)
ControllerTest (Tutorial\\Lesson5)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson4)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson5)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson7)

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.

Update

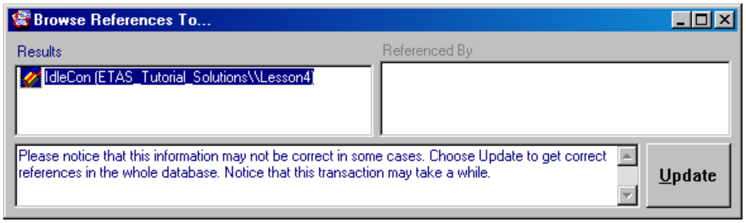
In the entire database, the `IdleCon` component is referenced by five projects with identical names (`ControllerTest`).

As described above, the reference to `IdleCon` is, with **Component → Replace References**, replaced by a reference to `IdleCon_1`. After the command has been executed, the "1 Database" field is unchanged, but in the element view it can be seen that the `ControllerTest` project, albeit using the old name, now references `IdleCon_1`. However, both components still exist in the database.



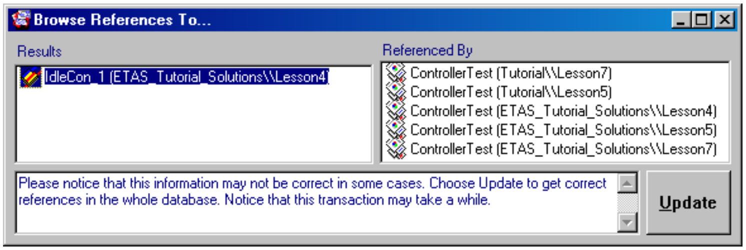
Name	Type	MaxSize	Scope	Kind
dT	dT		exported	Variable
IdleCon	IdleCon_1		exported	Variable
message	mesg[cont]		exported	Send Message
n	mesg[cont]		exported	Send Message

Checking both components with **Component → Show References** has the following result:



IdleCon (ETAS_Tutorial_Solutions\\Lesson4)

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.



IdleCon_1 (ETAS_Tutorial_Solutions\\Lesson4)

ControllerTest (Tutorial\\Lesson7)
ControllerTest (Tutorial\\Lesson5)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson4)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson5)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson7)

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.

There is no longer a reference to `IdleCon`, the five projects now reference `IdleCon_1`.

In ASCET 5.2 you can replace a database item and all the references to that item in other components or projects completely. This option is particularly important if you are working with a configuration management tool and need to merge divergent development streams for the same system.

Since every database item is known to the database only by its identity tag, the replacing item simply takes the place of another item by taking its identity tag. If, for example, your database has two objects A1 and A2, and you replace A1 with A2, the resulting object A2 gets the identifier and the references of A1 whilst keeping its functionality and the name A2.

To replace a database item:

- In the "1 Database" list of the Component Manager, select the item that is to replace the original item.
- Select **Component** → **Become another Item**.

  The "Select Item" window opens.
- Select the item you want to replace.
- Click **OK** to replace both the references to the second item and the item itself.
- In the "Confirm" window, confirm the command with **OK**.

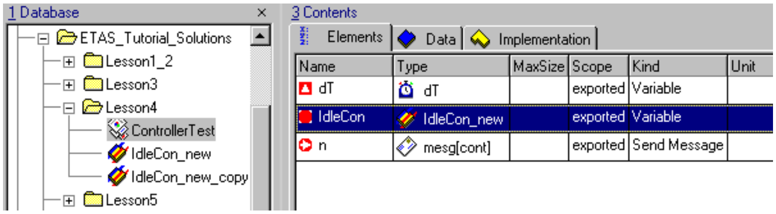
  The "Glue *<item>* with: *<item>*" window opens. All replacements are listed here.
- Confirm by clicking **OK**.

  The original instance of the item that replaces the second item retains its identifier and is renamed to `<item_name>_copy`.

Here is another example to clarify the way the command works. It is the same example as discussed above; this time however, the `IdleCon` component is completely replaced, not just the references to it.

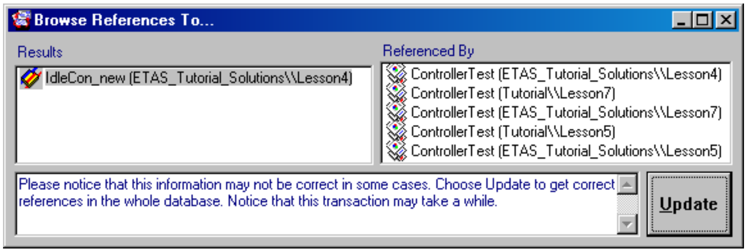
Before the **Component** → **Become another Item** command is executed, the `ControllerTest` project references the `IdleCon` component. Another project (`Project`) references the `IdleCon_new` component.

As described above, the **Component → Become another Item** command is used to replace the `IdleCon` component by `IdleCon_new`. After the command has been executed, the `IdleCon` component is deleted from the Component Manager. The "1 Database" list now contains the components `IdleCon_new` and `IdleCon_new_copy`.



Name	Type	MaxSize	Scope	Kind	Unit
dT	dT		exported	Variable	
IdleCon	IdleCon_new		exported	Variable	
n	mesg[cont]		exported	Send Message	

From the element view, it can be seen that `ControllerTest` (and the other project of that name) now references `IdleCon_new` under the old name. This component now has the identifier from the replaced component `IdleCon`.

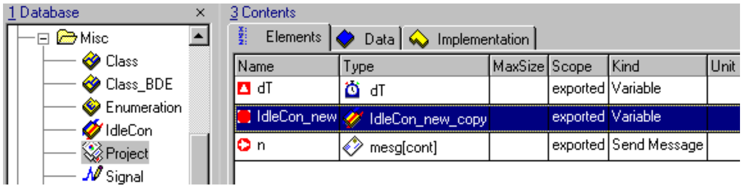


IdleCon_new (ETAS_Tutorial_Solutions\\Lesson4)

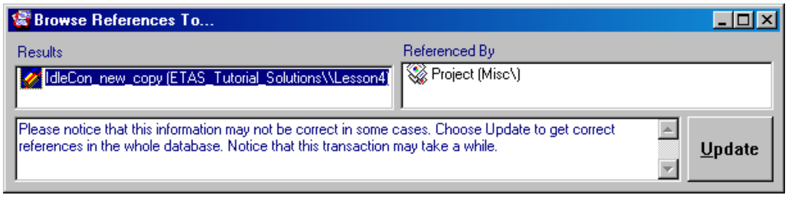
ControllerTest (ETAS_Tutorial_Solutions\\Lesson4)
ControllerTest (Tutorial\\Lesson7)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson7)
ControllerTest (Tutorial\\Lesson5)
ControllerTest (ETAS_Tutorial_Solutions\\Lesson5)

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.

The `IdleCon_new_copy` component contains the original instance of the replacing component and accordingly is referenced by the project `Project`.



Name	Type	MaxSize	Scope	Kind	Unit
dT	dT		exported	Variable	
IdleCon_new	IdleCon_new_copy		exported	Variable	
n	mesg[cont]		exported	Send Message	



IdleCon_new_copy (ETAS_Tutorial_Solutions\\Lesson4)

Project (Misc\)

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.

Editing

You can edit the layout of a component without first opening the respective component editor. The public interface of a component is declared in the layout editor.

To edit the layout of a component:

- Select a component.
- Select **Component** → **Edit Layout**.

  The layout editor opens for the selected component. It is described in "Editing the Layout of a Component" on page 511.

Every database item can have two types of text attached to it: comment text and notes. The comment text is entered in the "2 Comment" field in the Component Manager and is stored automatically. The notes for a database item are entered in a separate editor window. When documentation is generated automatically, comments, unlike notes, are not included.

To edit the notes for a database item:

- Select the database item with the notes you want to edit.

- Select **Component** → **Notes**.

  The notes editor opens for the selected database item. For details, see chapter 7.4 "Notes" on page 693.

Depending on the "Memory" attribute (see "Element Configuration" on page 448), variables and parameters are treated differently by the code generation. Only *volatile* elements are initialized automatically. *Non-volatile* data are not overwritten upon initialization.

To assign the *volatile* attribute to all variables:

To assign the *volatile* attribute to all variables in the database, proceed as follows:

- In the Component Manager, select **Tools** → **Database** → **Convert** → **Variables to Volatile**.

  All variables are now volatile and are thus initialized automatically.

To assign the *non-volatile* attribute to all parameters:

To assign the *non-volatile* attribute to all parameters in the database, proceed as follows:

- In the Component Manager, select **Tools** → **Database** → **Convert** → **Parameters to Nonvolatile**.

  All parameters are now non-volatile and are thus not overwritten upon initialization.

Find and Replace in C Code and ESDL Components

From within the Component Manager, you can search all C code and ESDL components for a character string. You can replace either an individual occurrence of the character string, every occurrence in a component or every occurrence in the entire database.

This function makes it easier to work with messages (or other global elements). These are linked via their names (see section "Interprocess Communication" in the ASCET reference manual); if one name is changed, all occurrences have to be changed. The search throughout the entire database means time-consuming manual searches are a thing of the past.

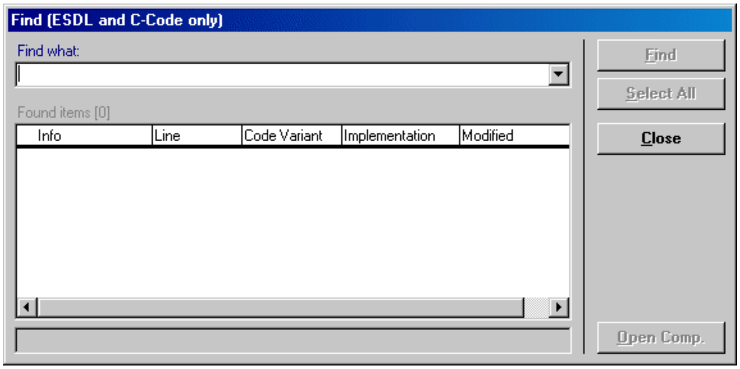
To find a character string:

- In the Component Manager, select **Edit** → **Find**

or

- press <CTRL> + <F>.

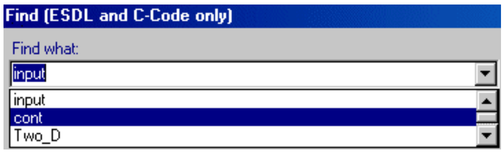
  The "Find (ESDL and C code only)" dialog box opens.



All buttons apart from **Close** are deactivated.

- Enter the character string you want to find in the "Find what" field.

  If you have performed other searches, you can select one of the previous search terms from the combo box of the field.



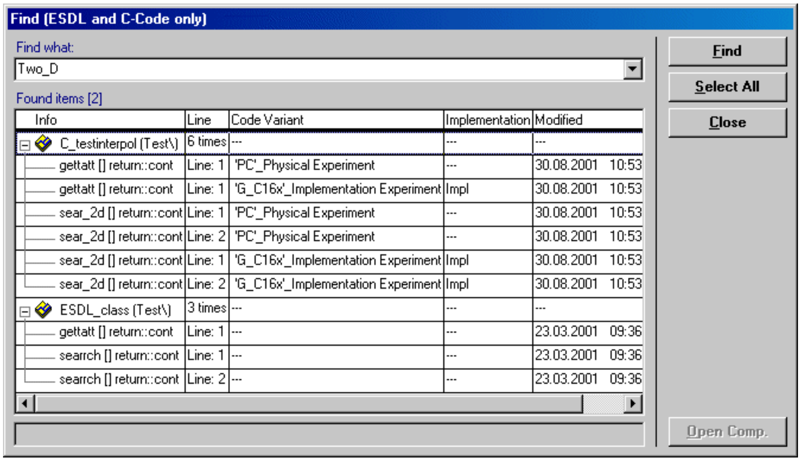
The **Find** button is now activated.

- Click **Find** to start the search.

  If you are searching the database for the first time, the search may well take a few minutes.

The search does not distinguish between upper and lower case. If, for example, you enter `cont`, `Cont` and `CONT` are found, too. The character string is also found if it is part of a longer word; for example, searching for `cont` also finds all occurrences of `Continuous`.

The results of the search are displayed in the "Found items" list; the **Select All** button is now activated. The number of components which contain the character string entered in the "Find what" field is shown in square brackets after the name of the list.



Find what: Two_D

Found items [2]

Info	Line	Code Variant	Implementation	Modified
⊟ C_testinterpol (Test\)	6 times	---	---	---
gettatt [] return::cont	Line: 1	'PC'_Physical Experiment	---	30.08.2001 10:53
gettatt [] return::cont	Line: 1	'G_C16x'_Implementation Experiment	Impl	30.08.2001 10:53
sear_2d [] return::cont	Line: 1	'PC'_Physical Experiment	---	30.08.2001 10:53
sear_2d [] return::cont	Line: 2	'PC'_Physical Experiment	---	30.08.2001 10:53
sear_2d [] return::cont	Line: 1	'G_C16x'_Implementation Experiment	Impl	30.08.2001 10:53
sear_2d [] return::cont	Line: 2	'G_C16x'_Implementation Experiment	Impl	30.08.2001 10:53
⊟ ESDL_class (Test\)	3 times	---	---	---
gettatt [] return::cont	Line: 1	---	---	23.03.2001 09:36
searrch [] return::cont	Line: 1	---	---	23.03.2001 09:36
searrch [] return::cont	Line: 2	---	---	23.03.2001 09:36

The "Found Items" list consists of five columns.

- The "Infos" column lists all components and all individual methods/ processes that contain the character string. The list of methods/processes is collapsed by default.
- The "Line" column specifies the number of occurrences of the character string for a component and the number of the line in which the character string occurs for individual methods/processes.
- The "Code Variant" column specifies which target and which experiment the code was specified for with C code components. The column is empty with ESDL components.
- The "Implementation" column contains the implementation used.
- The "Modified" column contains the date and time the component was last changed.

To view the search results:

- Click on the plus sign next to a component in the "Found Items" list

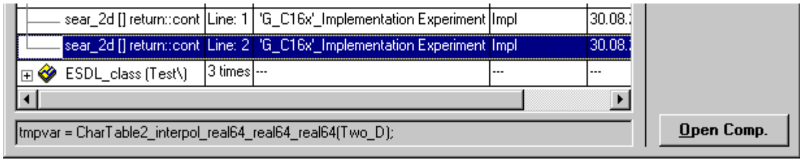
or

- double-click a component.

  The list of methods/processes opens.

- Click a line.

  The relevant code line is shown in the bottom bar of the window. The **Open Comp.** button is also activated.



sear_2d [] return::cont	Line: 1	'G_C16x'_Implementation Experiment	Impl	30.08.
sear_2d [] return::cont	Line: 2	'G_C16x'_Implementation Experiment	Impl	30.08.
ESDL_class (Test\)	3 times	---	---	---

```
tmpvar = CharTable2_interpol_real64_real64_real64(Two_D);
```

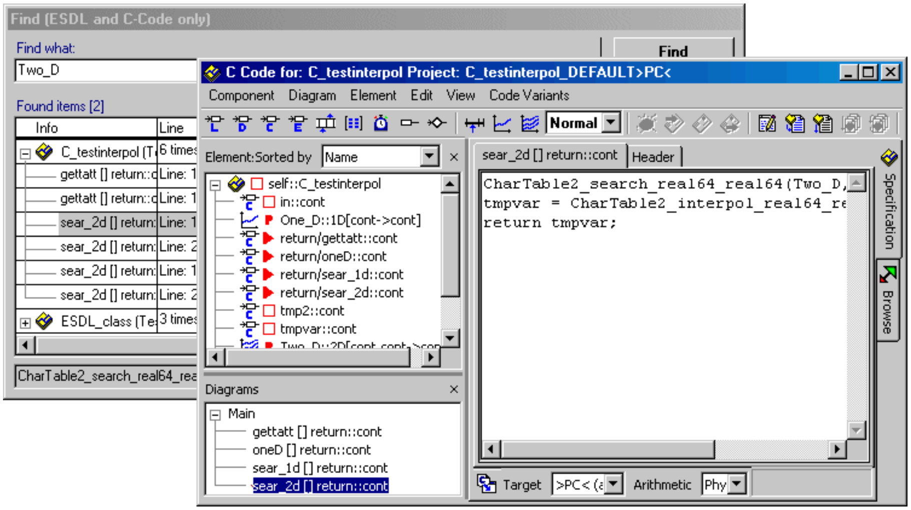
To open the component from the search window:

Proceed as follows to open the component from within the search window:

- Click the **Open Comp.** button.

  The editor for the component opens and the method/process is displayed. The line which contains the character string is highlighted.

  If it is a C code component, the code for the variant specified in the "Code Variant" and "Implementation" columns is displayed, recognizable by the contents of the combo box (see also chapter 4.3).



```
CharTable2_search_real64_real64(Two_D,
tmpvar = CharTable2_interpol_real64_re
return tmpvar;
```

To replace selected character strings:

Note

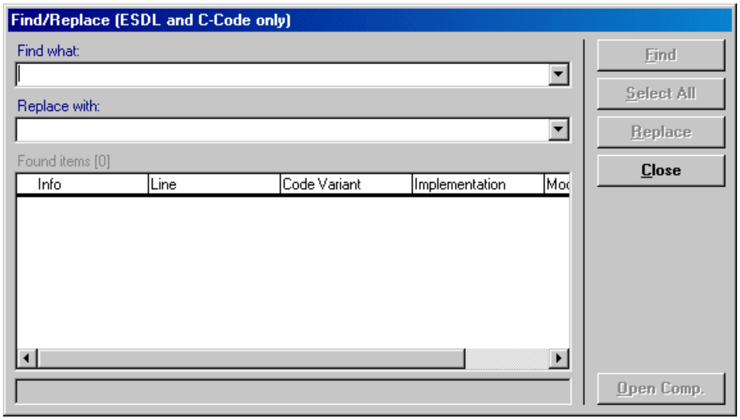
Be careful using the Replace function—there is ***no undo****!*

- In the Component Manager, select **Edit** → **Replace**

or

- press <CTRL> + <H>.

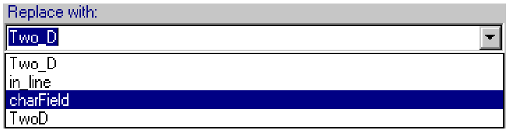
  The "Find/Replace (ESDL and C code only)" dialog box opens.



  Except for the "Replace with" field and the **Replace** button, this window is the same as the simple search window (cf. page 114).

- In the "Find what" field, enter the character string you want to replace.
- Enter the new text in the "Replace with" field.

  If you have performed other replace oprations, you can select one of the previous terms from the combo box of the field.



- Click **Find** to search for occurrences of the old character string.

Note

The order of the last two steps is irrelevant.

You can open the displayed components as described on page 115.
The **Replace** button is still deactivated.

- Find the component in which you want to replace the character string and expand the method/process list.
- Highlight one or more occurrences of the character string.

  The **Replace** button is now activated.
- Click **Replace**.

  The selected occurrences of the character string are replaced by the new text. The relevant lines disappear from the "Found Items" list.

To replace all character strings in one component:

Proceed as follows to replace all occurrences of the character string in a selected component:

- Find the character string you want to replace as described on page 116.
- In the "Find/Replace (ESDL and C code only)" window, enter the new text in the "Replace with" field.
- Highlight the component within which you want to replace the character string.

  With that, all occurrences of the character string in the component are selected.
- Click **Replace**.

  All occurrences of the character string in the selected component (for C code: including all implementations and targets) are replaced by the new text.

To replace all character strings in the database:

Proceed as follows to replace all occurrences of the character string:

- Find the character string you want to replace as described on page 116.
- In the "Find/Replace (ESDL and C code only)" window, enter the new text in the "Replace with" field.
- Click **Select All** to select all occurrences in all components.
- Click **Replace** to replace all occurrences in the entire database.

2.3.6 Managing Databases

When you start to work on a new project, it is helpful to create a new database. This operation is equivalent to changing to an empty dirctory on your computer. Then, you can create a customized folder structure, and create or import new items.

Note

You can read the name of the current database in the bottom bar of the Component Manager.

Basic Tasks

To create a new database:

Note

It is recommended to use several databases to keep the data volume small and comprehensible and to ensure optimal use of ASCET's performance.

- Select the **File** → **New Database** menu option

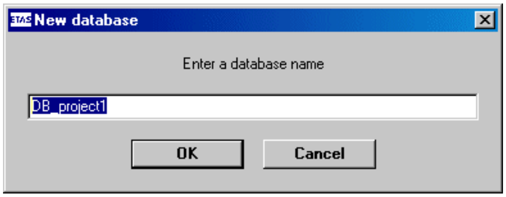
or

- click on the **New** button

or

- press <CTRL> + <N>.

  The "New database" window opens.



- Enter a name.
- Click on **OK**.

  The new database, empty except for the database name and the `Default` is opened. A subdirectory named as the database is created in the directory selected in the "Database Path" option (see page 41).
- Now proceed as described in "Managing Database Items" on page 74.

To load a database:

Note

You can also load databases from previous program versions (see "To convert an ASCET-SD 4.x database:" on page 128).

- Select **File** → **Open Database**

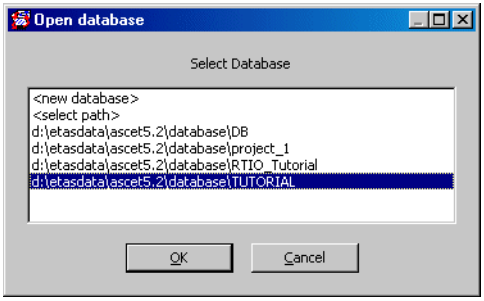
or

- click on the **Open** button

or

- press <CTRL> + <O>.

  The "Open database" window opens.



```
<new database>
<select path>
d:\etasdata\ascet5.2\database\DB
d:\etasdata\ascet5.2\database\project_1
d:\etasdata\ascet5.2\database\RTIO_Tutorial
d:\etasdata\ascet5.2\database\TUTORIAL
```

  The selection list includes all databases located in the default directory.

- Select the entry you want and click **OK**.

  If you want to load a database which is not located in the default directory, you can select the path required using the `<select path>` entry.

- Confirm by clicking **OK**.

  The database is loaded.

Note

You can also create a new database from this window. To do so, select the `<new database>` *entry.*

To save a database:

- Select **File** → **Save Database**

or

- click on the **Save** button

or

- press <CTRL> + <S>.

  All changes are saved in the database. Use this command regularly to save your database.

Note

*The user options (***Tools → Options***, "Options" node, cf. page 41) allow you to enable the Autocommit function and to enter the desired time interval. The database is also saved automatically when exiting the Component Manager.*

To save a database under a different name:

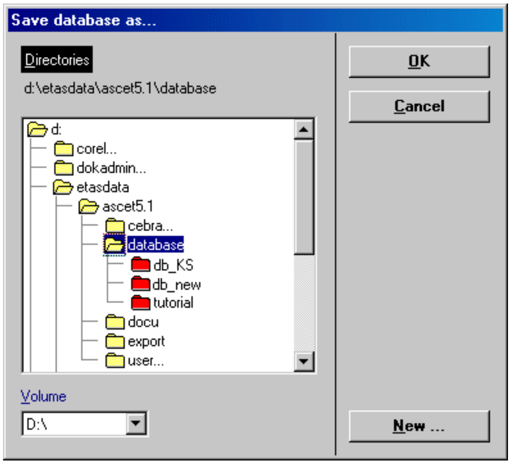
You can also save the database under any name. Thus you can, e.g., create a backup of an entire database and its supplementary files.

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

  The path selection window opens.

- Select the directory that will contain the database.

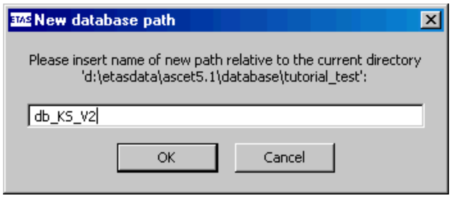
  You *must not* select a database.


Save database as...

Directories

d:\etasdata\ascet5.1\database

- Click on the **New** button to create a new, empty directory.

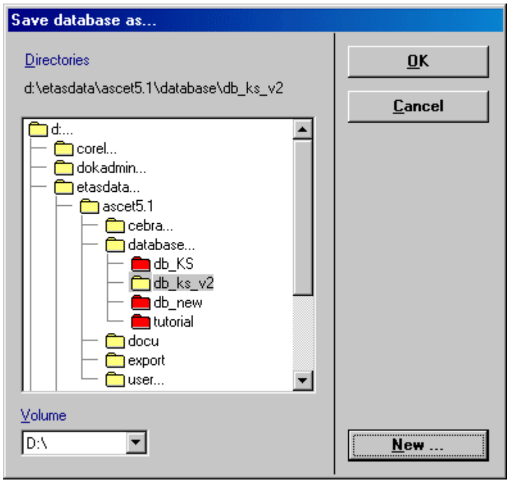


Please insert name of new path relative to the current directory 'd:\etasdata\ascet5.1\database\tutorial_test':

db_KS_V2

- Enter a name and click **OK**.

  The new directory is created, and selected as target directory for the backup copy. The color of the directory changes automatically after copying has finished.



d:\etasdata\ascet5.1\database\db_ks_v2

D:\

- In the path selection window, click **OK**.

Note

When you selected an existing directory, a warning is displayed that you will delete the entire content of that directory if you continue the backup procedure. You can cancel the procedure here.

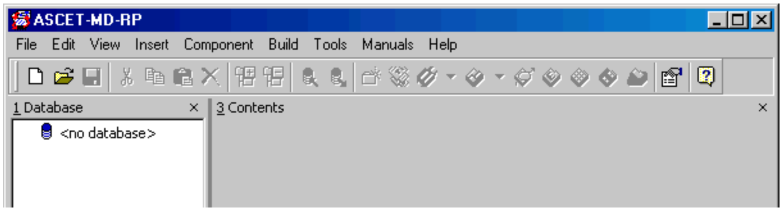
The content of the current database is copied, and the new database is created and loaded (see bottom bar in the Component Manager).

To close a database:

You can close the database you are working on without exiting ASCET.

- Select **File** → **Close Database**.

  The database is closed. The Component Manager remains open, but most buttons are deactivated.



To delete a database:

You *cannot* delete a database from within ASCET.

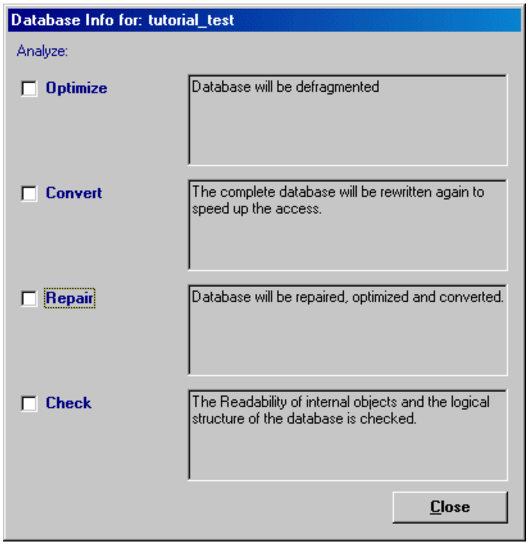
- If the database you want to delete is open in ASCET, close it.

  If you try to delete a database that is opened in ASCET, an error message occurs.
- In the Windows Explorer, select the directory (e.g., `..\ETASData\Ascet5.2`) that contains the database you want to delete.
- In the Windows Explorer, delete the subdirectory containing the database.

To optimize a database:

- In the Component Manager, select **Tools** → **Database** → **Performance Utilities.**

  The "Database Info" dialog window opens.



Database Info for: tutorial_test

Analyze:

- ☐ Optimize: Database will be defragmented
- ☐ Convert: The complete database will be rewritten again to speed up the access.
- ☐ Repair: Database will be repaired, optimized and converted.
- ☐ Check: The Readability of internal objects and the logical structure of the database is checked.

- Activate the **Optimize** option.

  This command is used to recombine the various scattered fragments in the database that result from intensive editing (defragmentation).

- Activate the **Convert** option.

  This command is used to cleanup the internal references between the data records ("X uses Y") after changes have been made to the database structure, thus improving access speeds.

In most cases, regular optimization will be enough to keep the performance of your database at an acceptable level. You should only consider converting the entire database if you encounter serious problems with performance.

Note

Converting a large database can be very time-consuming. It is recommended to run this as an overnight process.

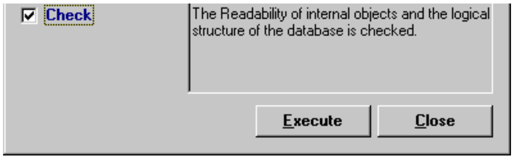
- Activate the **Repair** option.

  This command is used to rebuild the database from scratch. Damages of your database, e.g., destroyed references, are repaired.
  Repairing a database always includes the **Optimize** and **Convert** operations; the corresponding options are blocked.

- Activate the **Check** option.

  This command is used to verify the structures and references in the database while logging the results in the monitor window.

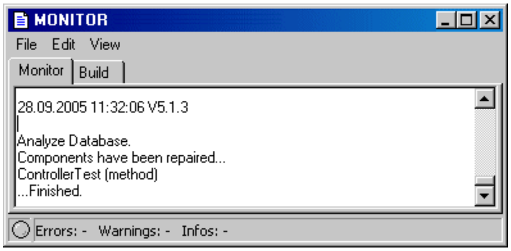
  Once you activated one option, the **Execute** button appears.



The Readability of internal objects and the logical structure of the database is checked.

- Click on the **Execute** button

  The selected database management program is started. The results are shown in the ASCET monitor window.



```
28.09.2005 11:32:06 V5.1.3
|
Analyze Database.
Components have been repaired...
ControllerTest (method)
...Finished.
```

Errors: -   Warnings: -   Infos: -

To compare two databases:

You can compare entire databases with each other. This is useful if you are working with different copies or development streams of the same database, or when working in development teams.

- In the Component Manager, select **Tools** → **Database** → **Compare Database**.

  The "Compare database" window opens.

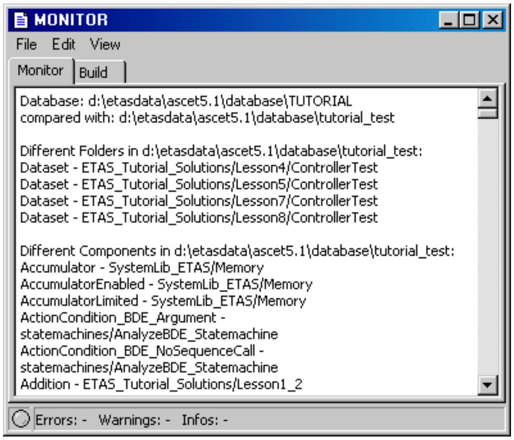
- Select the database that you want to compare with the current database.

- Click **OK** to start the comparison.

Note

Comparing large databases can be very time-consuming.

The databases are compared with each other. A list of items and projects in which the databases differ from each other is displayed in the monitor window.



```
Database: d:\etasdata\ascet5.1\database\TUTORIAL
compared with: d:\etasdata\ascet5.1\database\tutorial_test

Different Folders in d:\etasdata\ascet5.1\database\tutorial_test:
Dataset - ETAS_Tutorial_Solutions/Lesson4/ControllerTest
Dataset - ETAS_Tutorial_Solutions/Lesson5/ControllerTest
Dataset - ETAS_Tutorial_Solutions/Lesson7/ControllerTest
Dataset - ETAS_Tutorial_Solutions/Lesson8/ControllerTest

Different Components in d:\etasdata\ascet5.1\database\tutorial_test:
Accumulator - SystemLib_ETAS/Memory
AccumulatorEnabled - SystemLib_ETAS/Memory
AccumulatorLimited - SystemLib_ETAS/Memory
ActionCondition_BDE_Argument -
statemachines/AnalyzeBDE_Statemachine
ActionCondition_BDE_NoSequenceCall -
statemachines/AnalyzeBDE_Statemachine
Addition - ETAS_Tutorial_Solutions/Lesson1_2
```

Errors: - Warnings: - Infos: -

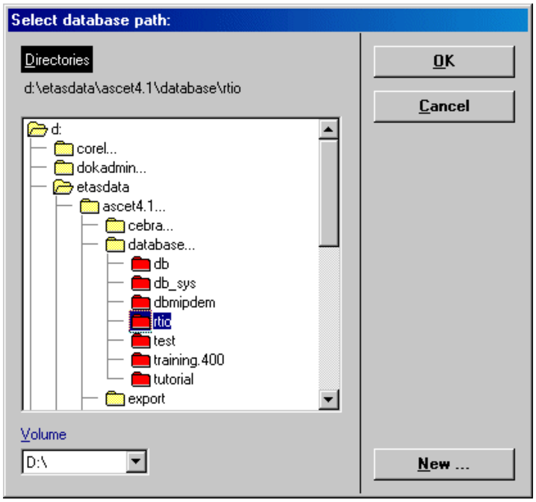
You can use existing databases from previous ASCET-SD versions with ASCET 5.2. These databases have to be converted, however, because the existing ASCET database format is not compatible with fortmats from previous ASCET-SD versions.

Conversion of existing databases is done automatically the first time you open a database that was created with ASCET-SD 4.x.

To convert an ASCET-SD 4.x database:

- Open the database as described in "To load a database:" on page 120.
- In the "Open Database" window, select the entry `<select path>` and click **OK**.

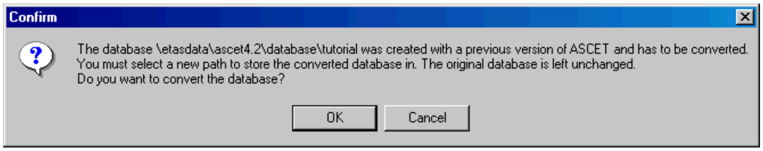
  The "Select database path" window opens.



- Select the directory that contains the old data-base.

- Confirm by clicking **OK**.

  The system prompts you to confirm the conversion of your database.



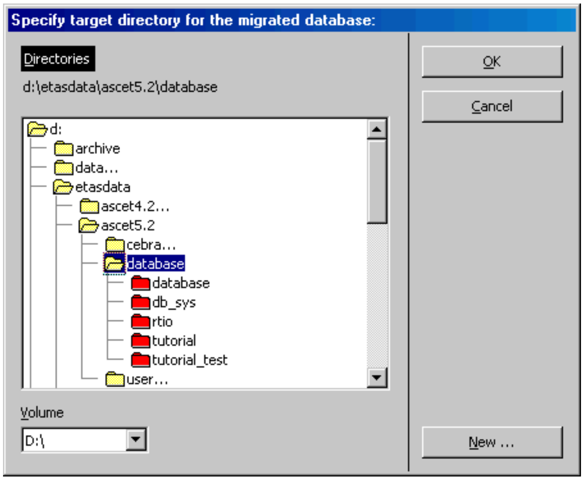
Confirm

The database \etasdata\ascet4.2\database\tutorial was created with a previous version of ASCET and has to be converted.
You must select a new path to store the converted database in. The original database is left unchanged.
Do you want to convert the database?

You can abort the conversion with **Cancel**.

- Click **OK** to confirm the conversion.

  The "Specify target directory for the migrated database" window opens. It works the same way as the "Select database path" window".



Specify target directory for the migrated database:

Directories

d:\etasdata\ascet5.2\database

```
d:
├ archive
├ data...
├ etasdata
│ ├ ascet4.2...
│ ├ ascet5.2
│ │ ├ cebra...
│ │ ├ database
│ │ │ ├ database
│ │ │ ├ db_sys
│ │ │ ├ rtio
│ │ │ ├ tutorial
│ │ │ └ tutorial_test
│ │ └ user...
```

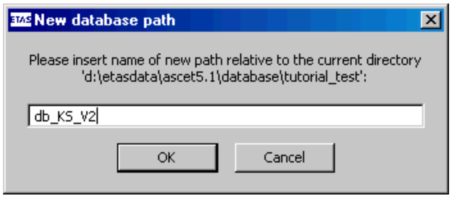
Volume

D:\

Note

When you select a non-empty directory here, a warning occurs. Confirm the warning and create a new, empty directory in the path selection window.

- Click on the **New** button to create a new, empty directory.



Please insert name of new path relative to the current directory 'd:\etasdata\ascet5.1\database\tutorial_test':

db_KS_V2

- Enter a name and click **OK**.

  The new directory is created, and selected as target directory for the conversion.

- Start the conversion procedure by clicking **OK** in the path selection window.

  The database is converted to the ASCET 5.2 format and written to the new directory. The duration of the conversion depends on the amount of data.
  The original old database remains unchanged.

Export files created with ASCET-SD earlier than version 4.0 cannot be imported into ASCET 5.2 because the current version offers no interfaces to the old database formats. When you try to open such database as described on page 128, you get the following error message:

```
This file is compatible with ASCET-SD V3.0 or earlier.
It cannot be imported directly. Convert it to a data-
base using your old version of ASCET-SD, then open
this database.
```

To convert a database from ASCET-SD prior to V4.0:

When you still have ASCET-SD 4.x, you can use very old databases with ASCET 5.2. Proceed as follows:

- Start ASCET-SD 4.x.

  These versions contain interfaces to the very old database formats.

- Open the old database.

  The database items are converted to the format of the corresponding ASCET-SD version.

- Close ASCET-SD 4.x.

- Start ASCET 5.2.
- Open the database you have just converted to ASCET-SD 4.x, as described on page 128.

ANSI C Conversion

In ASCET, ANSI C compliant naming is enforced for all the database items for versions above 2.x. When adding new items or renaming existing ones, you must use valid ANSI C identifiers.

The conversion of folder and item names in existing databases must be triggered explicitly after the database has been converted. All folder and item names are converted automatically, special characters are replaced as displayed in Tab. 2-4.

Special Characters	**Maps to ANSI C**
Ä, ä, Ö, ö, Ü, ü, ß	Ae, ae, Oe, oe, Ue, ue, ss
<space>, <dash>, <dot>	<underscore> for all items

Tab. 2-4 ANSI C character mapping

To convert a database to ANSI C:

- Open the database you want to convert.
- Select **Tools** → **Database** → **Convert** → **All Names to ANSI C**.

  The names of all folders, database items, methods, messages, variables, etc. are converted to ANSI-C using the mapping table displayed above for all special characters, punctuation marks and spaces.

Note

When converting large databases, you should consider compressing your database after conversion to speed up database access (see section "Database Maintenance Routines" on page 132).

Resolving Name Conflicts

In some cases, where item names in an existing version 2.x database are distinguished only by punctuation marks or spaces, the default mapping for conversion to ANSI C can lead to naming conflicts.

For example, the item names "Integrator I21" and "Integrator-I21" both map to "Integrator_I21". Since item names must be unique, the conversion algorithm would assign "Integrator_211" to the second item.

You can either accept this solution to naming conflicts when converting to ANSI C or edit the corresponding map file. The `mapping.ini` file is located in the ASCET directory and can be edited using any standard text editor.

Note

You cannot modify the mapping for spaces, they are always converted to underscores.

Database Maintenance Routines

ASCET provides tools for database repair, compression and backup. In addition, you can discard generated code for the entire database or force a new build and compare of the databases. This section explains the database utilities, which are accessed from the Component Manager.

For information on converting existing databases, see section "To convert an ASCET-SD 4.x database:" on page 128.

Generated Code

You can remove all the code generated during experiments from the database both to reduce the size of the database and to regenerate your code.

To discard the generated code stored in the database:

- In the Component Manager, select **Build** → **Clean All**.

  You are prompted to confirm the deletion of the generated code from the database.

- Confirm by clicking **OK**.

  All generated code is removed from the database.

During code generation, a make operation is performed, i.e. the code is generated and compiled for new or changed items only. Sometimes you may want to make sure that a build operation is performed, i.e. code is newly generated and compiled for all items. This does not remove previously generated code.

To force a new build during code generation:

- Select **Build** → **Touch All**.

  All components in the database are marked as changed (although no actual changes occured). Thus you ensure that next time, new code is generated and compiled.

Note

Forcing a new build can be very time-consuming if you are working on a large database.

Browsing the Database

You can browse the whole database for information on the relationship between items and elements, or operator implementations. For example, it is completely transparent in the Component Manager which items are used or referenced by other items. You can browse the database for these relationships using complex search criteria to display only the items in the database that fulfil the criteria.

When you browse the database ASCET always searches the entire database, regardless of which folders or items are currently selected. It is possible to use wildcard characters when specifying the search criteria, e.g. typing `*acc*` will find all database items and elements that have the string `acc` in their name. The search is not case-sensitive.

Searching for operator implementations (**Tools** → **Database** → **List Operator Implementations**) is described at the end of section "Operator Implementation" (page 504).

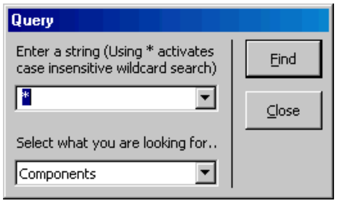
To browse the database ("Query" window):

- Open the Component Manager for the database you want to browse.
- Select **Edit** → **Query**

or

- press <CTRL> + <Q>.

  The "Query" window opens.



Enter a string (Using * activates case insensitive wildcard search)

*

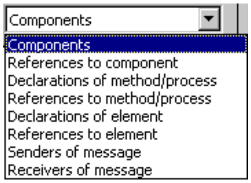
Select what you are looking for..

Components

- In the "Enter a string" field, enter a search term.

  The search term should be the name of a database entry, method, process, or element. The combo box of the field contains previous search terms, which you can select.

- In the "Select what you are looking for" combo box, select a search criterion (cf. page 136).



Components
Components
References to component
Declarations of method/process
References to method/process
Declarations of element
References to element
Senders of message
Receivers of message

- Click **Find** to start the search.

  The information found is shown in a dialog box.

To browse the database (toolbar):

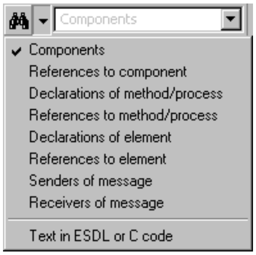
You can also browse the database via the Component Manager toolbar.

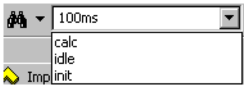
- In the Component Manager, click on the arrow button next to **Search - *<information type>***.

  A selection list opens. The currently selected search criterion (cf. page 136) is marked.

- Select a search criterion.

  The selected criterion is shown in pale font in the combo box.





- In the combo box, enter a new search term

or

- select a previous search term.

- Click on **Search - *<information type>*** to start the search.

  While the database is searched, the "Search *<objects>*" window is shown. In that window, click **Cancel** to abort the search.

  The resulting information is displayed in a dialog window.

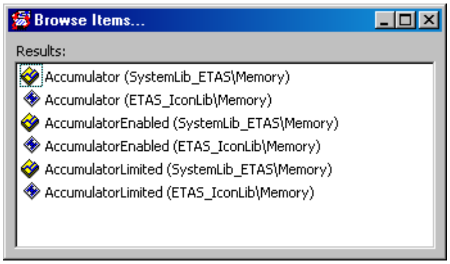
Note

If you selected the information type `Text in ESDL or C code`*,* ***Search - Text in ESDL or C code*** *does* ***not*** *open the results window. Instead, the "Find (ESDL and C-Code only)" window opens. Proceed as described in "To find a character string:" on page 113.*

The following types of information can be searched for:

Components:

This search criterion finds all the items in the database where the name matches the search string. The information is displayed in the following dialog window:

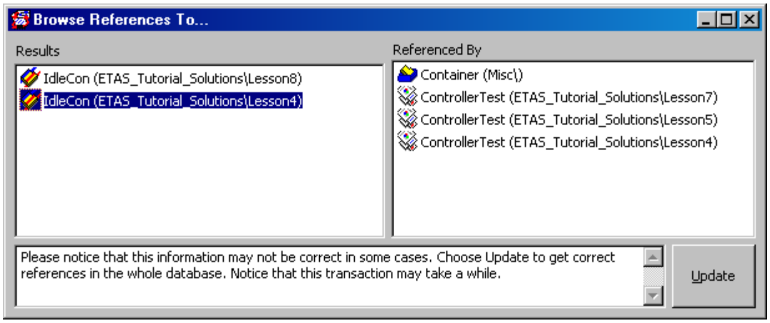


Accumulator (SystemLib_ETAS\Memory)
Accumulator (ETAS_IconLib\Memory)
AccumulatorEnabled (SystemLib_ETAS\Memory)
AccumulatorEnabled (ETAS_IconLib\Memory)
AccumulatorLimited (SystemLib_ETAS\Memory)
AccumulatorLimited (ETAS_IconLib\Memory)

The "Browse Items" dialog window shows the full names of the matching components together with their database path. If you click on a component name, the component is displayed and highlighted in the Component Manager.

References to component:

This search criterion finds all references to database components whose names match the search string. The information is displayed in the following dialog window.



IdleCon (ETAS_Tutorial_Solutions\Lesson8)
IdleCon (ETAS_Tutorial_Solutions\Lesson4)

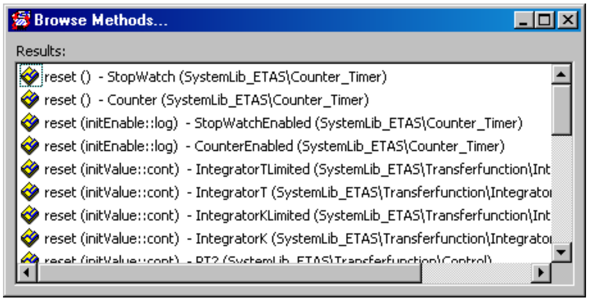
Container (Misc\)
ControllerTest (ETAS_Tutorial_Solutions\Lesson7)
ControllerTest (ETAS_Tutorial_Solutions\Lesson5)
ControllerTest (ETAS_Tutorial_Solutions\Lesson4)

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.

The items that match the search string are displayed in the "Results" pane of the dialog box. Click on an item in this pane to display its referencing items in the "Referenced By" pane. If you click on a component in either list, it is displayed and highlighted in the Component Manager.

Declaration of method/process:

This search criterion finds all methods or processes in the database which match the search string. These are displayed in the following dialog window:



Browse Methods...

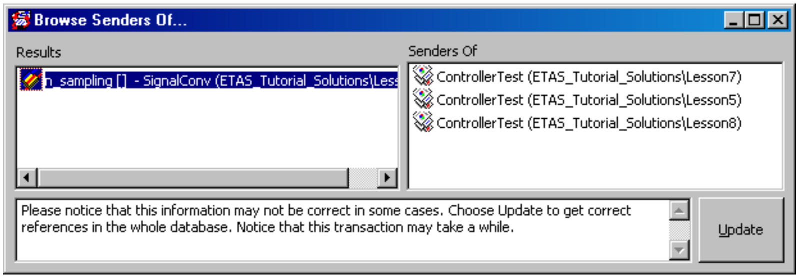
Results:

```
reset () - StopWatch (SystemLib_ETAS\Counter_Timer)
reset () - Counter (SystemLib_ETAS\Counter_Timer)
reset (initEnable::log) - StopWatchEnabled (SystemLib_ETAS\Counter_Timer)
reset (initEnable::log) - CounterEnabled (SystemLib_ETAS\Counter_Timer)
reset (initValue::cont) - IntegratorTLimited (SystemLib_ETAS\Transferfunction\Int
reset (initValue::cont) - IntegratorT (SystemLib_ETAS\Transferfunction\Integrato
reset (initValue::cont) - IntegratorKLimited (SystemLib_ETAS\Transferfunction\Int
reset (initValue::cont) - IntegratorK (SystemLib_ETAS\Transferfunction\Integrato
reset (initValue::cont) - PT2 (SystemLib_ETAS\Transferfunction\Control)
```

The "Browse Methods" dialog window shows the fully developed method selector and the database path of the defining component. Click on a method to display the defining component and highlight it in the Component Manager.

References to method/process:

This search criterion finds all references to methods or processes in the database whose names match the search string. These are displayed in the following dialog window:



Browse Senders Of...

Results

```
n_sampling [] - SignalConv (ETAS_Tutorial_Solutions\Less
```

Senders Of

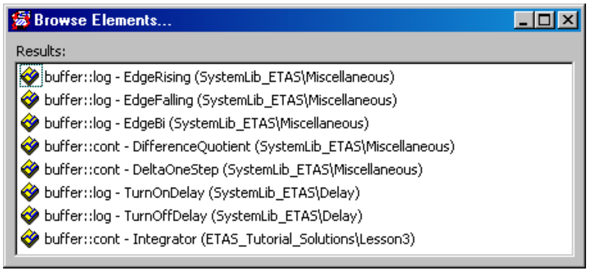
```
ControllerTest (ETAS_Tutorial_Solutions\Lesson7)
ControllerTest (ETAS_Tutorial_Solutions\Lesson5)
ControllerTest (ETAS_Tutorial_Solutions\Lesson8)
```

Please notice that this information may not be correct in some cases. Choose Update to get correct references in the whole database. Notice that this transaction may take a while.

The "Browse Senders Of" dialog window displays the items that match the search string in the "Results" pane. Method/process selectors are listed together with their defining components. Click on a method/process to list the components that call the method/process selected in the "Senders Of" pane of the dialog window. Click on an entry in either pane to display and highlight the relevant component in the Component Manager.

Declarations of element:

This search criterion finds all components that declare elements (e.g. variables or arguments) which match the search string. These are displayed in the following dialog window:


Browse Elements...

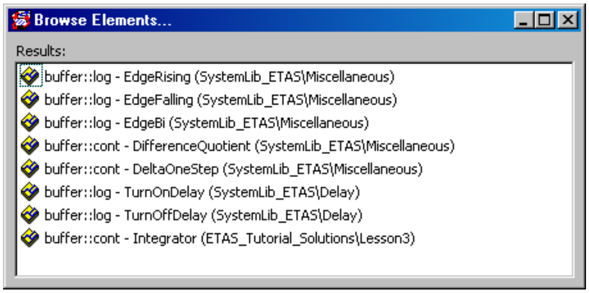
Results:

```
buffer::log - EdgeRising (SystemLib_ETAS\Miscellaneous)
buffer::log - EdgeFalling (SystemLib_ETAS\Miscellaneous)
buffer::log - EdgeBi (SystemLib_ETAS\Miscellaneous)
buffer::cont - DifferenceQuotient (SystemLib_ETAS\Miscellaneous)
buffer::cont - DeltaOneStep (SystemLib_ETAS\Miscellaneous)
buffer::log - TurnOnDelay (SystemLib_ETAS\Delay)
buffer::log - TurnOffDelay (SystemLib_ETAS\Delay)
buffer::cont - Integrator (ETAS_Tutorial_Solutions\Lesson3)
```

The "Browse Elements" dialog displays the elements together with the components in which they are defined and the database paths of these components. If you click on an element, the relevant component is displayed and highlighted in the Component Manager.

References to element:

This search criterion finds all components containing references to elements (e.g. variables or arguments) which match the search string. These are displayed in the following dialog window:


Browse Elements...

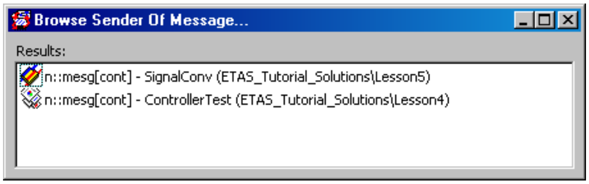
Results:

```
buffer::log - EdgeRising (SystemLib_ETAS\Miscellaneous)
buffer::log - EdgeFalling (SystemLib_ETAS\Miscellaneous)
buffer::log - EdgeBi (SystemLib_ETAS\Miscellaneous)
buffer::cont - DifferenceQuotient (SystemLib_ETAS\Miscellaneous)
buffer::cont - DeltaOneStep (SystemLib_ETAS\Miscellaneous)
buffer::log - TurnOnDelay (SystemLib_ETAS\Delay)
buffer::log - TurnOffDelay (SystemLib_ETAS\Delay)
buffer::cont - Integrator (ETAS_Tutorial_Solutions\Lesson3)
```

The "Browse Elements" dialog window shows the elements together with the component in which they are used, and the database path of that component. If you click on an element, the relevant component is displayed and highlighted in the Component Manager.

Senders of message:

This search criterion finds all modules or projects that send messages matching the search string. They are displayed in the following dialog window:



Browse Sender Of Message...

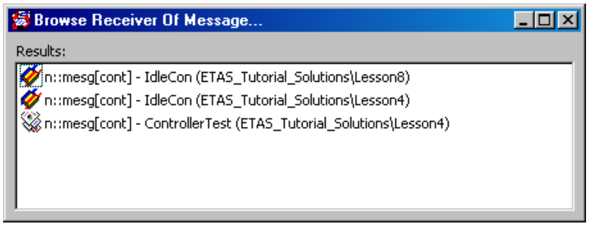
Results:

n::mesg[cont] - SignalConv (ETAS_Tutorial_Solutions\Lesson5)
n::mesg[cont] - ControllerTest (ETAS_Tutorial_Solutions\Lesson4)

The messages are shown together with the defining module/project. If you click on a message, the relevant module/project is displayed and highlighted in the Component Manager.

Receivers of message:

This search criterion works analogous to the previous one, but for received messages.



Browse Receiver Of Message...

Results:

n::mesg[cont] - IdleCon (ETAS_Tutorial_Solutions\Lesson8)
n::mesg[cont] - IdleCon (ETAS_Tutorial_Solutions\Lesson4)
n::mesg[cont] - ControllerTest (ETAS_Tutorial_Solutions\Lesson4)

2.3.7 Database Access

It is possible to adjust the access rights for each individual component, as well as those for entire folders. Access rights can be password protected, so that only users who know the password can change them. This ensures confidentiality when ASCET data is exchanged.

If, for instance, the access rights of a particular component are set to "execute only", that component can be used as a building block in other components, but it cannot be looked at, i.e. the algorithms it uses are protected from view.

The following access rights can be assigned:

Read: The database item can be looked at with the appropriate item editor.

Write: Items can be added to the database and deleted. Items can be edited in the appropriate editor.

Calibration: Items can be calibrated in experiments.

Execute: Items can be experimented on. This also goes for referenced items, i.e. if a user has no execute rights to an item, they cannot use it within another item, and then experiment with that.

Code Generation: This option allows code generation even if write access is not set. If read access is set, the component can be viewed, but not modified; experimentation is possible.

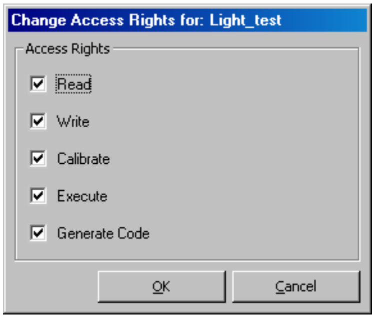
The most recent changes always apply. If, for instance, access rights for a single component are changed, and then later the rights for the entire folder are changed, those changes overrule the ones made earlier to the component.

Your access rights to the item selected are displayed in the bottom bar of the Component Manager.

To change the access rights of a folder or item:

- In the "1 Database" list of the Component Manager, select the desired folder or item.
- Select **Component** → **Access Rights**.

  The "Overwrite Access Rights for" dialog opens. It displays the current access rights for the item selected.



Change Access Rights for: Light_test

Access Rights

- [✓] Read
- [✓] Write
- [✓] Calibrate
- [✓] Execute
- [✓] Generate Code

- Activate options to grant access rights.
- Deactivate options to revoke access rights.
- Click **OK** to confirm your changes.

Access rights can be specified for any database item. However, only limited modifications are permitted if password protection has been activated. Password protection can be activated separately for each root folder in the database. If password protection is activated, users can modify access rights only if they know the correct password.

Note

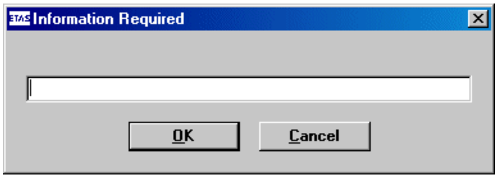
If password protection is active, you must enter the password for every change to the access rights. This can be quite cumbersome if you want to modify more than just a few items. Therefore, it is more efficient to specify access rights first and then activate password protection.

To activate password protection:

- In the "1 Database" list of the Component Manager, select a top-level folder.
- Select **Component** → **Password**.

  You are prompted to confirm the activation of their password protection.
- In the "Confirm" window, confirm with **Yes**.

  The "Information Required" window opens.



- Enter a password that is at least six characters long and click **OK**.

  A verification window for the password opens.
- Retype the password and click **OK** to activate password protection.

  Users now have to enter the password whenever they want to modify access rights in this folder.

You can deactivate password protection by entering the password again.

To deactivate password protection:

- In the "1 Database" list of the Component Manager, select the top-level folder whose password protection you want to deactivate.
- Select **Component** → **Password**.

  The "Information Required" window opens.
- Type in the password and click **OK**.

  You are asked to confirm the deactivation.
- Confirm with **Yes**.
- Password protection is deactivated.

2.4 The ASCET Monitor Window

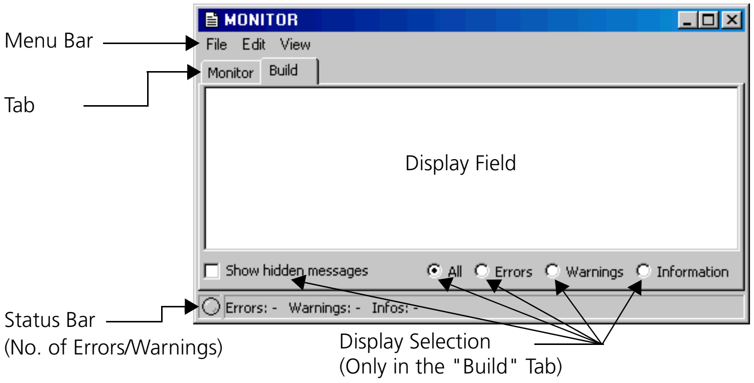
The results of various operations are displayed in the ASCET monitor window in two tabs.

The "Monitor" tab shows the results of the following operations:

- code generation, experimenting (see chapter 4.1.10, chapter 4.2.5, chapter 4.5.4 and chapter 4.8.10)
- analysis of diagrams (see page 258 and chapter 4.4.3)
- database optimizations (see page 125)
- export and import processes (see chapter 2.3.3 and chapter 2.3.4)
- search for operator implementations (see page 504)

The "Build" tab shows the results of the following operations:

- code generation, experimenting
- analysis of diagrams



Menu Functions:

- **File**
  - *Open (<CTRL> + <O>)*

    Opens an existing log file. The file always opens in the "Monitor" tab even if the "Build" tab is currently displayed. The content of the "Monitor" tab is overwritten during loading.
  - *Save (<CTRL> + <S>)*

    Saves the content of the tab displayed in the file `ASCET_Monitor.log` ("Monitor" tab) or `CodeGeneration.log` ("Build" tab).
  - *Save As*

    Saves the content of the tab displayed under a name of your choice.
  - *Show in Editor*

    Opens the log file of the current tab in a text editor. The text editor can be selected in the "External Tools" node of the ASCET option window, (see page 59).
  - *Reposition*

    Repositions the monitor window to the original position and size on the screen.
  - *Exit (<ALT> + <F4>)*

    Closes the ASCET monitor window.

- **Edit**
  (also available as context menu in the "Monitor" tab)

Note

This menu is only available in the "Monitor" tab.

  - *Cut (*<CTRL> + <X>*)*

    Cuts the selected text out of the display field.
    This item is the only one that can be undone with <CTRL> + <Z>.
  - *Copy (*<CTRL> + <C>*)*

    Copies text from the display field into the clipboard.
  - *Paste (*<CTRL> + <V>*)*

    Inserts text from the clipboard into the display field.
    This command can be undone with <CTRL> + <Z>.
  - *Select All (*<CTRL> + <A>*)*

    Selects the entire content of the display field.
  - *Find/Replace (*<CTRL> + <F>*)*

    Finds/replaces text in the display field.
  - *Clear (*<CTRL> + <R>*)*

    Deletes the content of the display field.
- **View**

Note

This menu is only available in the "Build" tab.

  - *Collapse All*

    The display is collapsed as far as possible.
  - *Expand All*

    The display is expanded completely.

Context Menus.

- "Monitor" tab

  The context menu in the "Monitor" tab contains the same functions as the **Edit** menu.
- "Build" tab

- *Open*

  Opens the editor for the component which caused the message.
- *File Out*

  Saves the content of the tab under a name of your choice.
- *Hide*

  Hides all messages of a selected type.
- *Promote to warning*

  Promotes information to the status of warning.
- *Promote to error*

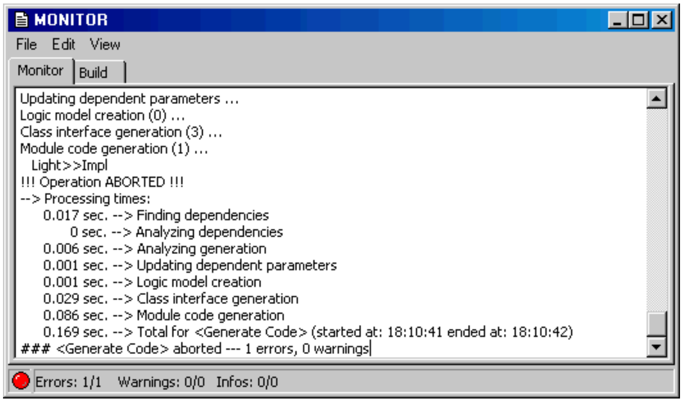
  Promotes information or a warning to the status of error message.
- *Revoke Promotion*

  Revokes a promotion and reestablishes the original type.
- *Settings*

  Opens the "CodeGen Message Configuration" window (see page 154).

2.4.1 "Monitor" Tab

The results of various operations are displayed in the "Monitor" tab. As an example, the screenshot below shows the results of the code generation and compilation of a state machine. Older log texts are not overwritten; the new log texts are simply added to the end.



```
Updating dependent parameters ...
Logic model creation (0) ...
Class interface generation (3) ...
Module code generation (1) ...
  Light>>Impl
!!! Operation ABORTED !!!
--> Processing times:
    0.017 sec. --> Finding dependencies
         0 sec. --> Analyzing dependencies
    0.006 sec. --> Analyzing generation
    0.001 sec. --> Updating dependent parameters
    0.001 sec. --> Logic model creation
    0.029 sec. --> Class interface generation
    0.086 sec. --> Module code generation
    0.169 sec. --> Total for <Generate Code> (started at: 18:10:41 ended at: 18:10:42)
### <Generate Code> aborted --- 1 errors, 0 warnings
```

To save the content of the "Monitor" tab:

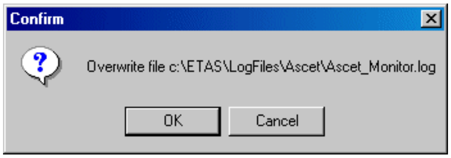
If you want to save the content of the "Monitor" tab, proceed as follows.

- Select **File** → **Save**

or

- press <CTRL> + <S> to save the text in the `Ascet_Monitor.log` file in the `ETAS\LogFiles\Ascet` directory.

  If this file exists, you are prompted to confirm that the existing file will be overwritten.



Overwrite file c:\ETAS\LogFiles\Ascet\Ascet_Monitor.log

- Confirm with **OK**.

  The content of the "Monitor" tab is saved.

To save the content of the "Monitor" tab under a different name:

- Select **File** → **Save As** to save the log file under a name and path of your choice.

  A file selector window opens.

- Select a path and a file name and click **Save**.

  The content of the "Monitor" tab is written to the relevant file.

If you do not want to save the entire text of the "Monitor" tab, you can delete, copy and add sections in the monitor window in the same way as you can with any other text. The **Edit** menu contains the items **Cut**, **Copy** and **Paste** for this purpose. You can also find and replace text strings using **Edit** → **Find/ Replace**.

Note

Remember that there is no undo function when you are editing.

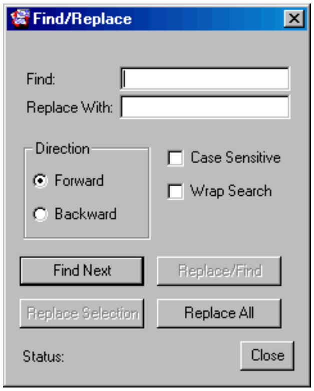
To find/replace in the "Monitor" tab:

- Select **Edit** → **Find/Replace**

or

- press <CTRL> + <F>.

  The "Find/Replace" window opens.



- In the "Find" box, enter the search string.
- In the "Replace With" box, enter the text that is to replace the text entered in the "Find" box.
- Select the direction under "Direction".
- Activate **Case Sensitive** if the search is to take upper and lower case into consideration.
- Activate **Wrap Search** if the search is to be continued at the beginning once the end has been reached in the search direction.
- Click **Find Next** to find the next occurrence of the search string.
- Click **Replace Selection** to replace the selected occurrence.
- Click **Replace/Find** to replace the selected occurrence and search for the next one.
- Click **Replace All** to replace the search string each time it occurs.
- Close the "Find/Replace" window with **Close**.

If you want to delete the entire content of the tab, e.g. before an operation whose result you want to save, proceed as follows:

To clear the tab content:

- Select **Edit** → **Clear**

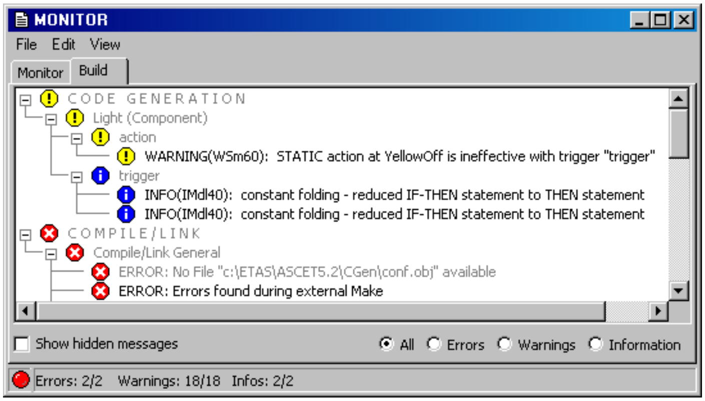
or

- press <CTRL> + <R>.

  The entire text in the "Monitor" tab is cleared.

2.4.2 "Build" Tab

The "Build" tab displays the results of code generation and compilation as well as of the analysis of diagrams. The screenshot below shows the results of the same code generation and compilation as in the "Monitor" tab. Unlike the "Monitor" tab, only the results of the last operation are displayed here.



```
CODE GENERATION
  Light (Component)
    action
      WARNING(WSm60): STATIC action at YellowOff is ineffective with trigger "trigger"
    trigger
      INFO(IMdl40): constant folding - reduced IF-THEN statement to THEN statement
      INFO(IMdl40): constant folding - reduced IF-THEN statement to THEN statement
COMPILE/LINK
  Compile/Link General
    ERROR: No File "c:\ETAS\ASCET5.2\CGen\conf.obj" available
    ERROR: Errors found during external Make

Errors: 2/2   Warnings: 18/18   Infos: 2/2
```

Three types of message issued in the named operations are displayed: Errors (red dot), warnings (yellow dot) and information (blue dot). The different types of message are distinguished by abbreviations, e.g. `WSm60` or `IMdl40`. The display for every operation is ordered in a tree structure according to components and methods/processes.

To make it easier to localize the problem areas, the errors and warnings displayed are also links to the problem areas.

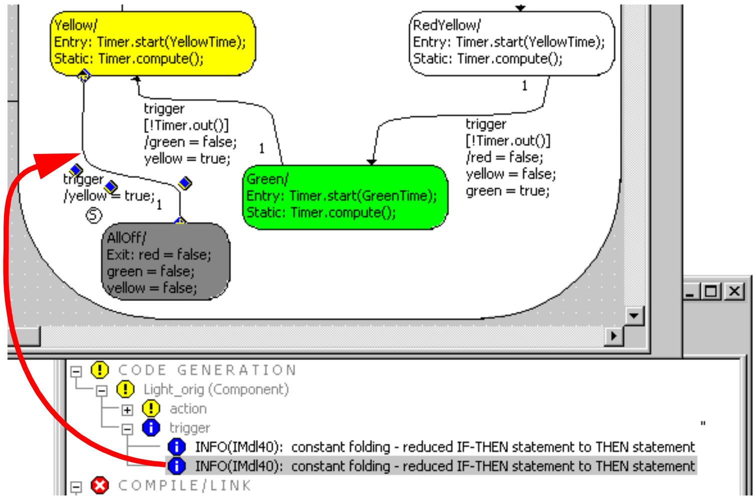
To display the cause of the message:

- Double-click a message in the "Build" tab

or

- right-click a message and select **Open** from the context menu.

  The editor for the component concerned opens and the relevant point is activated.



```
CODE GENERATION
  Light_orig (Component)
    action
    trigger
      INFO(IMdl40): constant folding - reduced IF-THEN statement to THEN statement
      INFO(IMdl40): constant folding - reduced IF-THEN statement to THEN statement
COMPILE/LINK
```

In the example shown, the transition from the status `AllOff` to the status `Yellow` caused the warning.

When compiler or linker errors have occurred, a text window or text editor opens showing the relevant generated file. The use of a text editor is controlled in the "External Tools" node of the ASCET option window (see page 59).

To save the content of the "Build" tab:

If you want to save the content of the "Build" tab, proceed as follows.

- Select **File** → **Save**

or

- press <CTRL> + <S> to save the log text in the `CodeGeneration.log` file in the `ETAS\LogFiles\Ascet` directory.

  Unlike in the "Monitor" tab, an existing file of this name is simply overwritten *without any warning*.

  The content displayed in the "Build" tab is written to the relevant file.

Note

*Hidden messages (see page 151) are **not** saved.*

To save the content of the "Build" tab under a different name:

- Select **File Out** from the context menu

or

- select **File** → **Save As**

or

- right-click in the tab and select **File Out** from the context menu to save the content of the "Build" tab under a name and path of your choice.

  A file selector window opens.

- Select a path and a file name and click **Save**.

  The content displayed in the "Build" tab is written to the relevant file.

Configuring Messages in the "Build" Tab

There are different ways of configuring the display of messages from the "Build" tab. Other possible configurations are described in the section "Configuring Messages in the "CodeGen Message Configuration" Window" on page 154. Your configurations are saved in the options of the current project (when working with components: of the default project) and can be accessed from there (see page 414).

Showing/hiding messages. You can hide messages of the same type to keep the display clearer or to analyze one specific type of message.

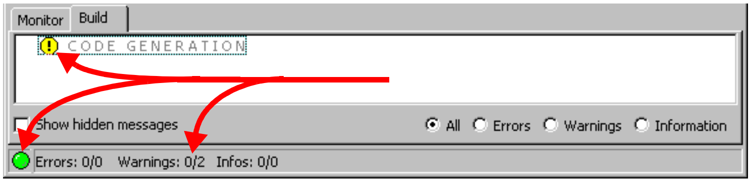
Note

Error messages cannot be hidden (even if they are promoted warnings or promoted information).

To hide messages:

- Right-click the type of message you want to hide in the "Build" tab.
- Select **Hide** from the context menu.

  All messages of the selected type are hidden. The number of messages displayed in the footer of the monitor window is adjusted.
  If you hide all error messages and warnings, a green dot is shown at the left of the footer.
  The number as well as the colored icons in the display field indicate there are hidden messages.



CODE GENERATION

Errors: 0/0 Warnings: 0/2 Infos: 0/0

To show all hidden messages:

You can display all hidden messages again.

- Activate **Show hidden messages** in the "Build" tab.

  All hidden messages are displayed.

Note

There is ***no*** *option to show hidden messages of a single type. This can only take place in the "CodeGen Message Configuration" window (see page 154).*

Promoting messages. Information and warnings do not interrupt the code generation process. The relevant context of a project may require a stricter interpretation for some information or warnings which is why it is possible to promote information or warnings of one type. The table below shows the possible promotions.

↓ Promote to →	**Information**	**Warning**	**Error message**
Information		Possible	Possible
Warning	---		Possible
Error message	---	---	

The type of promoted message remains which is why promoting does not affect showing/hiding messages.

If a hidden message is promoted, it is displayed again.

To promote information and warnings:

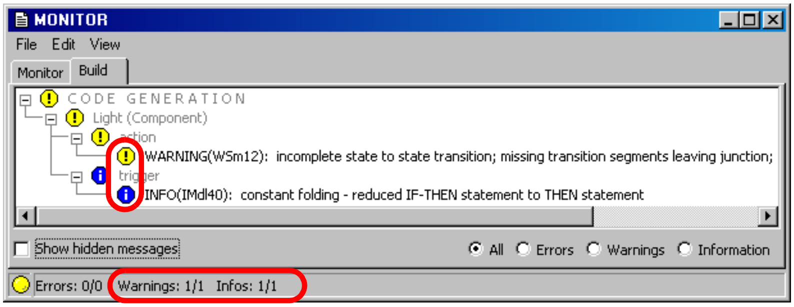
- Right-click the type of message you want to promote in the "Build" tab.
- Select **Promote to warning** from the context menu

or

- select **Promote to error** from the context menu.

  The messages of the type selected are counted as warnings or error messages from the next time code is generated and shown accordingly in the "Build" tab.

The following figures are intended to clarify promoting. The first figure shows a warning and an information message as they are issued by code generation.

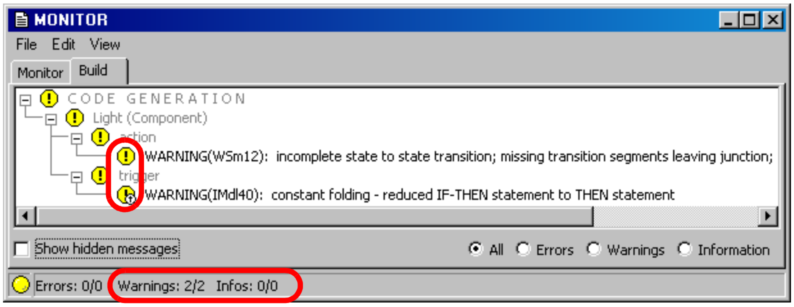


WARNING(WSm12): incomplete state to state transition; missing transition segments leaving junction;

INFO(IMdl40): constant folding - reduced IF-THEN statement to THEN statement

Errors: 0/0 Warnings: 1/1 Infos: 1/1

In the second figure, the information has been promoted to a warning. The type remains the same – `IMdl40` – but the message is assigned the icon for warnings with an indication that this warning is promoted information: . In the "Build" tab, two warnings are displayed and counted.


MONITOR

File Edit View

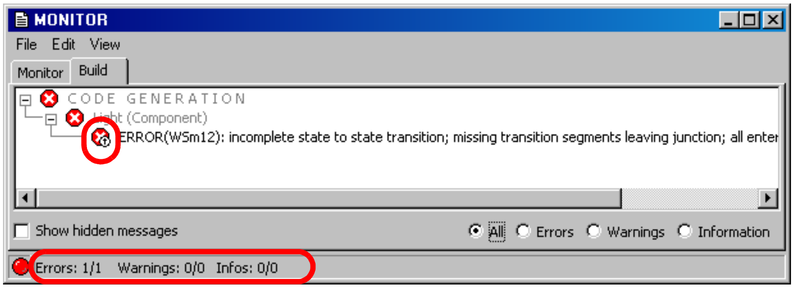
Monitor Build

```
CODE GENERATION
  Light (Component)
    action
      WARNING(WSm12): incomplete state to state transition; missing transition segments leaving junction;
    trigger
      WARNING(IMdl40): constant folding - reduced IF-THEN statement to THEN statement
```

☐ Show hidden messages   (●) All ( ) Errors ( ) Warnings ( ) Information

Errors: 0/0 Warnings: 2/2 Infos: 0/0

In the third figure, the warning has been promoted to an error message. The type remains the same – `WSm12` – but the message is assigned the icon for error messages with an indication that this is a promoted message: . Promotion to an error message causes code generation to abort at this point; it does not reach the stage of the second message. This is why only one error message is displayed in this case in the "Build" tab.


MONITOR

File Edit View

Monitor Build

```
CODE GENERATION
  Light (Component)
    ERROR(WSm12): incomplete state to state transition; missing transition segments leaving junction; all enter
```

☐ Show hidden messages   (●) All ( ) Errors ( ) Warnings ( ) Information

Errors: 1/1 Warnings: 0/0 Infos: 0/0

To revoke a promotion:

If a promoted message is displayed in the "Build" tab, you can revoke the promotion.

- Right-click a promoted message in the "Build" tab.

- Select **Revoke Promotion** from the context menu.

  At the next code generation, messages of this type are again displayed in their original form.

Configuring Messages in the "CodeGen Message Configuration" Window

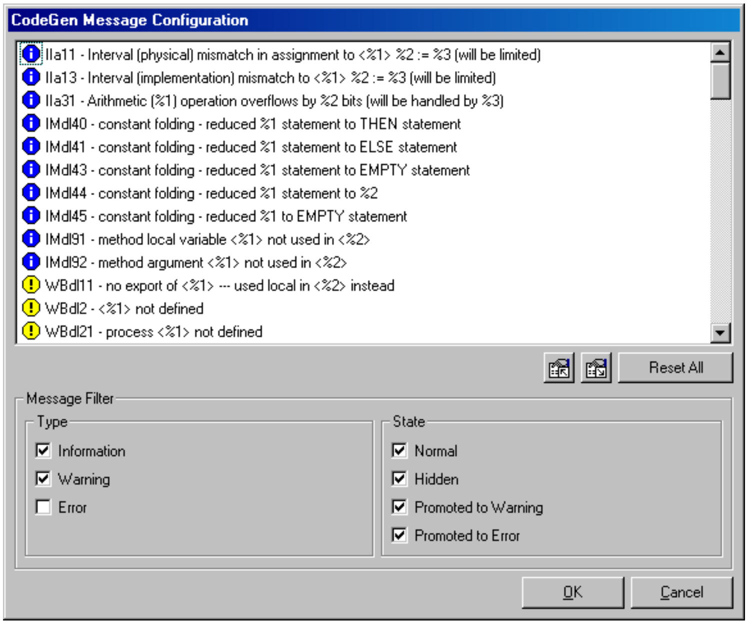
The "CodeGen Message Configuration" window contains other configuration possibilities. You can promote or hide not only the messages currently displayed but all existing messages, you can revoke promotions and redisplay hidden messages of one type.

To open the "CodeGen Message Configuration" window:

From the monitor window, you can only open the window if at least one message is displayed in the "Build" tab. For details of how to get to the "CodeGen Message Configuration" window via the project options, refer to the section ""Build" Node" on page 412.

- Right-click the "Build" tab.
- Select **Settings** from the context menu.

  The "CodeGen Message Configuration" window opens.



CodeGen Message Configuration

```
IIa11 - Interval (physical) mismatch in assignment to <%1> %2 := %3 (will be limited)
IIa13 - Interval (implementation) mismatch to <%1> %2 := %3 (will be limited)
IIa31 - Arithmetic (%1) operation overflows by %2 bits (will be handled by %3)
IMdl40 - constant folding - reduced %1 statement to THEN statement
IMdl41 - constant folding - reduced %1 statement to ELSE statement
IMdl43 - constant folding - reduced %1 statement to EMPTY statement
IMdl44 - constant folding - reduced %1 statement to %2
IMdl45 - constant folding - reduced %1 to EMPTY statement
IMdl91 - method local variable <%1> not used in <%2>
IMdl92 - method argument <%1> not used in <%2>
WBdl11 - no export of <%1> --- used local in <%2> instead
WBdl2 - <%1> not defined
WBdl21 - process <%1> not defined
```

Message Filter

Type	State
☑ Information	☑ Normal
☑ Warning	☑ Hidden
☐ Error	☑ Promoted to Warning
	☑ Promoted to Error

The window contains the following elements:

- List field

  This field can display all information, warnings and error messages. You control the display using the options in the "Message Filter" box. The list is sorted alphabetically according to type.

- **Import Message Configuration from XML File** and **Export Message Configuration to XML File** buttons

  Start import and export of message configuration.

- **Reset All** button

  Resets all messages to the state originally specified by the system.

- "Message Filter" box

  You use the options in this box to set up the display in the list field.

  – "Type" box with the options **Information**, **Warning** and **Error**

    These options enable/disable the display of the relevant type of message globally. The following options specify the details.

  – **Normal**

    Enables/disables the display of unprocessed messages (not hidden, not promoted) of the selected types.

  – **Hidden**

    Enables/disables the display of hidden messages of the selected types.

  – **Promoted to Warning** and **Promoted to Error**

    Enables/disables the display of messages promoted to warnings or error messages of the selected types.

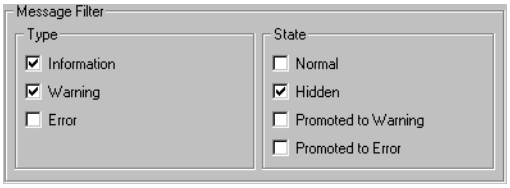
To set up the display:

- Activate the options of the types of message you want to display in the list field in the "CodeGen Message Configuration" window.

  Information, **Warning** and **Error** are at your disposal.

- Define the display using **Normal**, **Hidden** and **Promoted to Warning/Error**.

  This enables you to search for specific groups of messages (only unprocessed information, no hidden messages, promoted warnings etc.). The following setting ensures, for example, that only hidden messages are displayed.


Message Filter

Type
- [x] Information
- [x] Warning
- [ ] Error

State
- [ ] Normal
- [x] Hidden
- [ ] Promoted to Warning
- [ ] Promoted to Error

To hide messages in the monitor window:

Note

Error messages cannot be hidden (even if they are promoted warnings or promoted information).

- Select one or more messages you want to hide in the "CodeGen Message Configuration" window.
- Select **Hide** from the context menu.

  Promoted messages are indicated with paler icons in the list field, e.g. these:
- Close the "CodeGen Message Configuration" window with **OK**.

  All messages of the selected type are hidden in the "Build" tab of the monitor window.

  The number of messages displayed in the footer of the monitor window is adjusted. If you hide all error messages and warnings, a green dot is shown at the left of the footer. The number as well as the colored icons in the display field indicate there are hidden messages.

To show hidden messages:

- Select one or more hidden messages you want to show in the "CodeGen Message Configuration" window.

  To make it easier to find the hidden messages, it may be helpful to disable **Normal** and **Promoted to ***.
- Select **Show** from the context menu.
- Close the "CodeGen Message Configuration" window with **OK**.

  All messages of the selected type are shown in the "Build" tab of the monitor window.

To promote information and warnings:

- Select one or more messages you want to promote in the "CodeGen Message Configuration" window.
- Select **Promote to warning** from the context menu

or

- select **Promote to error** from the context menu
- Close the "CodeGen Message Configuration" window with **OK**.

  The messages of the type selected are counted as warnings or error messages from the next time code is generated and shown accordingly in the "Build" tab.
  Promoted messages are indicated with special icons in the list field:

  information promoted to a warning

  information promoted to an error message

  warning promoted to an error message

To revoke a promotion:

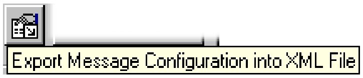
- Select one or more messages whose promotion you want to revoke in the "CodeGen Message Configuration" window.
- Select **Revoke Promotion** from the context menu.
- Close the "CodeGen Message Configuration" window with **OK**.

  Messages of these types are once again displayed in their original form during the next code generation in the "Build" tab.

You can export your configuration of hidden and promoted messages to an XML file and you can import this kind of configuration from an XML file.

To export settings:

- Open the "CodeGen Message Configuration" window.



- Click the **Export Options of selected Node into XML File** button.

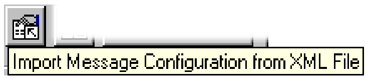
  A confirmation window opens.
- If you do not want the confirmation window to be displayed in future, disable **Show next time**. (see page 45.)
- Confirm the saving of your changes with **OK**.

  The Windows file selection dialog box opens. `*.xml` is specified as format.
- Set path and name of the export file.
- Click **Save**.

  The hidden and promoted messages are written to the XML file specified.

To import settings:

- Open the "CodeGen Message Configuration" window.



- Click the **Import Message Configuration from XML File** button.

  The Windows file selection dialog box opens. `*.xml` is specified as format.

- Select the XML file which contains the configuration required.
- Click **Open**.

  The hidden and promoted messages contained in the XML file specified are imported and displayed in the "CodeGen Message Configuration" window and in the monitor window accordingly.

3 Adding User-Defined Functions

It is possible to define your own functions to make it easier to work with ASCET add-ons (e.g. version management) and external programs. These are:

- your own menus in different windows,
- an autostart action (executed when ASCET is started),
- a shutdown action (executed when ASCET is shut down).

These are specified in `*.ini` files which are either stored in a subdirectory of the ASCET installation, `ETAS\Ascet5.2\Executer`, or of the data directory, `ETASData\Ascet5.2\Executer`. The `Executer` subdirectory can in turn contain subdirectories.

Note

To keep the content of the `Executer` *subdirectory in case of an update or re-installation of ASCET, you have to place it below the data directory (*`ETASData\Ascet5.2`*).*

The `*.ini` files are only read when ASCET is started; later changes are only effective after a restart. The syntax of these files corresponds to the Windows `*.ini` format.

Each `*.ini` file consists of one or more sections whose names are in square brackets. The following sections are available:

- **`[<NAME>]`** — defines a new menu item. You can select any name you wish, but each name can only occur once in an `*.ini` file. Please refer to section 3.1 "Defining Menu Items".
- **`[AUTOSTART]`** — defines the autostart action. Please refer to section 3.2 "Defining an Autostart Action".
- **`[SHUTDOWN]`** — defines the shutdown action. Please refer to section 3.3 "Defining a Shutdown Action".

3.1 Defining Menu Items

The section of the `*.ini` file which defines a menu item is as follows:

```
[<Name>]
WINDOW=<window name>
MENU=~<menu name>
ITEM=~<menu item name>
DESCRIPTION="<text>"
SEPARATOR=<separator>
```

```
FILE=<filename>.txt
```

Note

*The terms to the left of the equals sign **must** be written in block caps.*

The variables have the following significance:

- `<Name>` is the name of the section. The name must be unique within the `*.ini` file. *No* difference is made between upper and lower case; for example `[Input]` and `[INPUT]` are the same.

  Value: any, but unique
- `<window name>` is the name of the window in which the menu is generated.

  Value: see Tab. 3-1 on page 163
- `<menu name>` is the name of the menu which is generated in the specified window.

  Value: any
- `<menu item name>` is the name of the menu item which is generated in the `<menu name>` menu.

  Value: any
- `<text>` is a description of the menu item which is only required for documentation purposes.

  Value: any
- `<separator>` specifies whether a separator is generated before or after the menu item. The line can be omitted if there is to be no separator (`Nothing`).

  Value: `Before / After / Nothing`

- `<filename>` – the script file `<filename>.txt` is executed when the menu item is selected. The structure of these files is explained in section 3.4 "Structure of the Script Files".

  Value: any

Name of the Window	**ASCET Window**
`mainwindow` or `databasebrowser`	Component Manager
`projecteditor`	Project editor
`blockdiagrameditor`	Block diagram editor
`smeditor`	State machine editor
`esdleditor`	ESDL editor
`ccodeeditor`	C code editor
`conttimeblockdiagrameditor`	Block diagram editor (CT blocks)
`conttimeesdleditor`	ESD editor (CT blocks)
`conttimeccodeditor`	C code editor (CT blocks)
`booleantableeditor`	Boolean table editor
`conditionaltableeditor`	Conditional table editor
`datadialog`	"Data for ..." window
`impldialog`	Implementation editor
`onlinesimulation`	Online experiment environment
`offlinesimulation`	Offline experiment environment
`intecriobackanimation`	Experiment environment for back animation with INTECRIO
`incabackanimation`	Experiment environment for back animation with INCA

Tab. 3-1 Names of the Windows into which Menu Items can be Inserted

If several `*.ini` files are created, they are evaluated in alphabetical order. The menu items which are defined in a file called `asd_menu.ini` are further up the relevant menu than items in the same menu which are defined in `my_menu.ini`.
Within the same `*.ini` file, the menu items are created in the order in which they are defined.

To define menu items:

- Create a new `*.ini` file

or

- Open an existing `*.ini` file.
- Enter a name for the section, e.g.

  `[Input]`.

  Make sure that the name is used only once in the file.
- Enter the window in which the menu is to be generated, e.g.,

  `WINDOW=databasebrowser`.
- Enter a name for the menu, e.g.

  `MENU=~My Menu`.
- Enter the name the menu item should have, e.g.

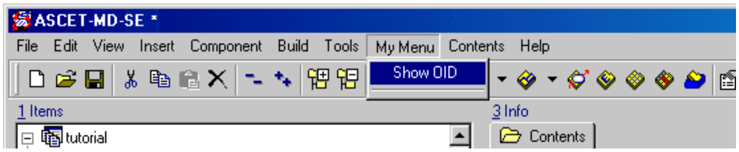
  `ITEM=~Show OID`.
- Enter a description, e.g.,

  `DESCRIPTION=show object IDs.`
- Enter where you want a separator,e.g.,

  `SEPARATOR=After`.
- Enter the script file, e.g.,

  `FILE=show.txt`.

The next time ASCET is started, **My Menu** will be generated in the Component Manager.


ASCET-MD-SE *
File Edit View Insert Component Build Tools My Menu Contents Help
Show OID
1 Items
3 Info
tutorial
Contents

If you want to define several menu items in the same or in different ASCET windows, you have to create such a section for each individual menu item. Constructions like the following do *not* work; the second definition is ignored.

```
[EXPORT]
WINDOW=databasebrowser
MENU=~My Menu
ITEM=~Show OID
DESCRIPTION=show object IDs
```

```
FILE=show.txt
ITEM=~open editor
DESCRIPTION=opens external text editor
FILE=openeditor.txt
```

The sections can either be in the same or in separate `*.ini` files. The advantage of using separate files is that it is easier to control the order of the menu items in a menu using the file names.

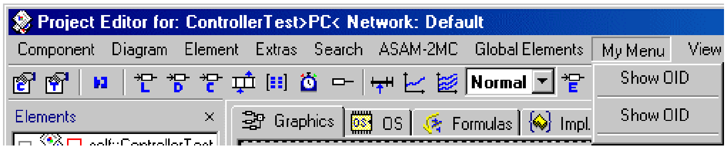
If you use one `*.ini` file for several items, make sure that each name is only used once. If a name occurs a second time, the first definition is executed a second time regardless of what the second definition is.

An example: the following sections are to be used to generate the menu item **My Menu → Show OID** in the project editor and the menu item **My Menu → Open Editor** in the Component Manager.

```
[Input]
WINDOW=projecteditor
MENU=~My Menu
ITEM=~Show OID
DESCRIPTION=show object IDs
SEPARATOR=After
FILE=show.txt

[INPUT]
WINDOW=databasebrowser
MENU=~My Menu
ITEM=~Open Editor
DESCRIPTION=open external text editor
SEPARATOR=After
FILE=openeditor.txt
```

As there is no distinction between upper and lower case, the names of the sections are identical and the menu item **Show OID** is generated twice under **My Menu** in the project editor.



3.2 Defining an Autostart Action

An autostart action is defined in the `[AUTOSTART]` section of the `*.ini` file. The section can be anywhere in the `*.ini` file, but can occur *only once*, even if several `*.ini` files are used. It contains only one line, `FILE="<file-name>"`.

Example:

```
[AUTOSTART]
FILE=start.txt
```

The `start.txt` script file is executed at the start of ASCET.

The autostart function is used to execute configurations for file operations (e.g. adapting templates for code generation, deleting directories) when ASCET is started. This results in a unique, reproducible situation. An editor or any other tool can also be invoked to execute the necessary preparatory work and settings for the session. A message (s. page 169) can provide information on the settings made.

An example of this kind of script file:

```
EXECUTE c:\Programme\TextPad 4\TextPad.exe
        C:\ETAS\Ascet5.2\target\c16x\codegen.ini
OBJECT message: "codegen.ini is updated"
```

You cannot access COM-API with the autostart function as the automation server is not started at this time.

3.3 Defining a Shutdown Action

A shutdown action is defined in the `[SHUTDOWN]` section of the `*.ini` file. The section can be anywhere in the `*.ini` file; it contains only one line, `FILE=<filename>`.

Example:

```
[SHUTDOWN]
FILE=shutdown.txt
```

When ASCET is shut down, the `shutdown.txt` script file is executed.

You cannot access COM-API with the shutdown function as the automation server has already been shut down at this time.

3.4 Structure of the Script Files

The script files (line `File =...` in the `*.ini` file) are written as text files. Nine keywords are used to specify the various actions:

`EXECUTE`, `NOWAIT`, `OBJECT`, `SELECTOR`, `MENUITEM`, `FILE`, `FORK`, `SEND`, `Post`.

The use of these keywords is explained in the following sections.

Unlike with the `*.ini` files, changes to the script files are effective as soon as the relevant menu item is next selected.

3.4.1 EXECUTE

The keyword `EXECUTE` is used to start an external program or a batch file. As long as the external program is running, the execution of the script file is interrupted.

Examples:

- `EXECUTE C:\ETAS\Ascet5.2\import.bat` invokes the batch file `import.bat` in the directory `C:\ETAS\Ascet5.2`.
- `EXECUTE C:\PVCS\nt\PVCSvmnt.exe` starts the version management program PVCS.

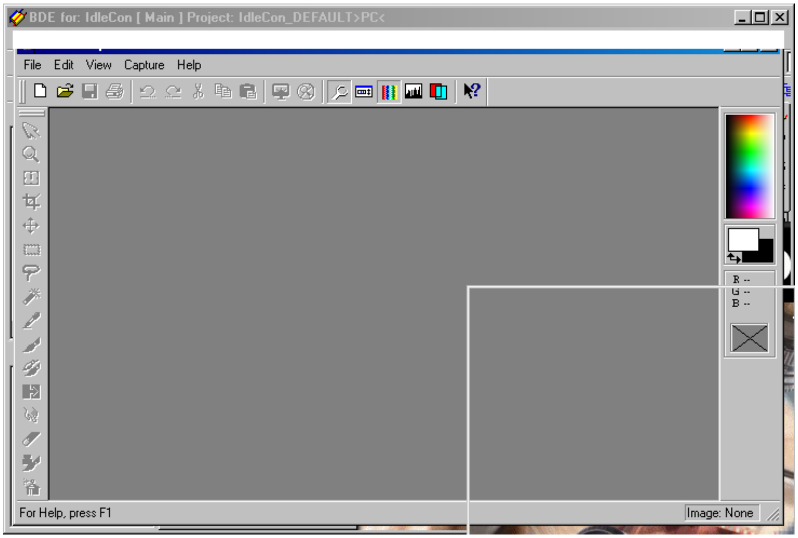
The interruption of the execution can have undesired results. If, for example, a menu item **My Menu** → **open PSP** is added to make it easier to create screenshots, the following happens:

- The menu item invokes the script file.

  ```
  OBJECT popWindow: BDE
  EXECUTE c:\Programme\Paint Shop Pro 5\psp.exe
  ```

- The first line results in the block diagram editor popping up; the second opens the image processing program.

But because the execution of the script file is interrupted, the block diagram editor is not updated, which results in the following screen.


BDE for: IdleCon [Main] Project: IdleCon_DEFAULT>PC<
File Edit View Capture Help
For Help, press F1
Image: None

Swapping the lines around does not solve the problem because this would result in the block diagram editor popping up after the image processing program has been closed.

3.4.2 NOWAIT

The keyword `NOWAIT` is used in exactly the same way as `EXECUTE`. The execution of the script file is not, however, interrupted here.

Example:

The example is the same as at the end of the previous section, but `NOWAIT` and not `EXECUTE` is now used in the script file.

```
OBJECT popWindow: BDE
NOWAIT c:\Programme\Paint Shop Pro 5\psp.exe
```

Again the first line results in the block diagram editor popping up; the second opens the image processing program. As the execution of the script file is not interrupted here, the block diagram editor is updated immediately and the screenshot can be taken.

3.4.3 OBJECT

The keyword `OBJECT` can be used together with a range of identifiers for different commands.

- **`help: <editor>`**

  Displays the `exe_hlp.txt` help file in the specified editor.

  Example: `OBJECT help: notepad` displays the help in the Notepad editor.

  Note

  The blank after the colon is mandatory for all identifiers. The command cannot be executed if the colon is forgotten.

- **`log: <editor>`**

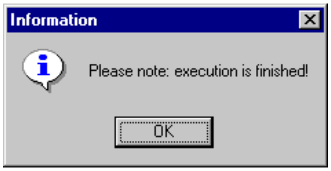
  Displays the `exe_log.txt` log file in the specified editor.

  Example: `OBJECT log: notepad`

- **`message: <text>`**

  Opens a window with a message for the user.

  Example: `OBJECT message: "Please note: execution is finished!"` results in the following window:

  Please note: execution is finished!

- **`popWindow: <title>`**

  The ASCET window with the specified name pops up. `<title>` does not have to be the complete name; part of the name specified in the title bar is sufficient.

  Example: `OBJECT popWindow: Database` – the Component Manager pops up.

- **`generatePopWindowCommandFile: <filename>`**

  Creates a file with the specified name in which the correct `popWindow` command is generated for the ASCET window currently active. The required file ending *must* be specified in `<filename>`, path specifications are optional.

  Example: `OBJECT generatePopWindowCommandFile: pop.txt` generates the `pop.txt` file with the following contents when invoked from the Component Manager (only ASCET-MD is installed):

  ```
  OBJECT popWindow: "ASCET-MD"
  ```

- **`wait: <n>`**

  Interrupts execution of the script file for `n` seconds.

  Example: the lines

  ```
  OBJECT wait: 5
  EXECUTE C:\PVCS\nt\PVCSvmnt.exe
  ```

  result in a 5-second delay before PVCS is started.

- **`windows: <editor>`**

  Displays a list (`exe_win.txt`) of the Smalltalk names of all open ASCET windows in the specified editor.

  Example: `OBJECT windows: notepad` displays the list in Notepad.

3.4.4 SELECTOR

The keyword `SELECTOR` is used to invoke a range of predefined commands in the Component Manager or the specification editors.

Example: Output of object IDs of selected components

`SELECTOR executerFileOutSelectedElementsTo: "<filename>"` creates a file which contains the object IDs. The required file ending *must* be specified in `<filename>`; path specifications are optional.

Note

If you make path specifications in `<filename>`, make sure that all subdirectories exist. Otherwise the file cannot be created.

A menu item is added to the Component Manager which invokes the following script file.

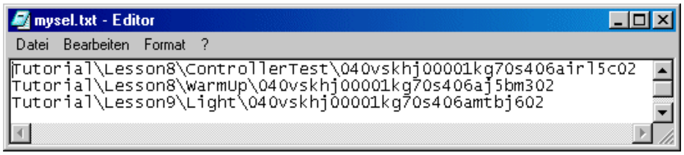
```
SELECTOR executerFileOutSelectedElementsTo:
     ".\Executer\mysel.txt"
```

```
EXECUTE Notepad .\Executer\mysel.txt
```

```
MENUITEM View>Update
```

The `ControllerTest`, `WarmUp` and `Light` components are selected and then the menu item is invoked. The following takes place:

- The first line (`SELECTOR ...`) creates the `mysel.txt` file in the `Executer` subdirectory of the installation directory.
- The second line (`EXECUTE ...`) opens the newly created file in Notepad.



```
Tutorial\Lesson8\ControllerTest\040vskhj00001kg70s406airl5c02
Tutorial\Lesson8\WarmUp\040vskhj00001kg70s406aj5bm302
Tutorial\Lesson9\Light\040vskhj00001kg70s406amtbj602
```

- The third line (`MENUITEM ...`) updates the Component Manager once Notepad has been closed.

Available Commands:

- `executerCollapseAll`

  Function: collapses the element list in the "1 Database" field.

  Use: in the Component Manager

  Syntax:

  ```
  SELECTOR executerCollapseAll
  ```

- `executerExpandCollapseSelectedElement`

  Function: expands and collapses the selected directory in the "1 Database" field. (Subdirectories are collapsed but not expanded again.)

  Use: in the Component Manager

  Syntax:

  ```
  SELECTOR executerExpandCollapseSelectedElement
  ```

- `executerDeselectElements`

  Function: undoes the component or element selection.

  Use: in the Component Manager as well as the specification editors

  Syntax:

  ```
  SELECTOR executerDeselectElements
  ```

- `executerFileOutSelectedDiagrams`

  Function: writes all diagrams and hierarchies of the selected component as well as the components in it to the ASCET installation directory as image files.
  This command has no effect on ESDL or C-Code components.

  Use: in the Component Manager

  Syntax:

  ```
  SELECTOR executerFileOutSelectedDiagrams
  ```

- `executerFileOutSelectedElementsTo:`

  Function: writes the object IDs of the selected elements into the specified file.

  Use: in the Component Manager as well as the specification editors

  Syntax[1]:

  ```
  SELECTOR executerFileOutSelectedElementsTo:
           "<filename>"
  ```

- `executerSelectElementFromString:`

  Function: selects the first visible element in the "1 Database" list (Component Manager) or "Elements" list (specification editors) whose name starts with the search string <name>.
  Elements in collapsed folders or components that correspond to the search string and all following corresponding visible elements are ignored.

  Use: in the Component Manager as well as the specification editors

  Syntax[2]:

1. `<filename>` *must* contain the file ending; path specifications are optional.
2. Upper and lower case are not taken into consideration in the search.

```
SELECTOR executerSelectElementFromString:
         "<name>"
```

- `executerFileOutDiagrams`

  Function: writes all diagrams and hierarchies of the edited component to the ASCET installation directory as image files.
  This command has no effect on ESDL or C code components.

  Use: in the specification editors

  Syntax:

  ```
  SELECTOR executerFileOutDiagrams
  ```
- `executerFileOutSelectedDiagramTo:`

  Function: writes the path name of the selected diagram of the edited component into the specified file. Images of the selected diagram and the hierarchies contained in it are additionally generated for block diagrams in the ASCET installation directory.

  Use: in the specification editors

  Syntax[1]:

  ```
  SELECTOR executerFileOutSelectedDiagramTo:
    "<filename>"
  ```
- `executerSelectDiagramFromString:`

  Function: selects the diagram that contains the search string `<name>` but without loading it.

  Use: in the specification editors

  Syntax[1]:

  ```
  SELECTOR executerSelectDiagramFromString:
    "<name>"
  ```

- `executerSelectPage:`

  Function: Opens the specified project editor tab.

  Use: in the project editor

  Syntax:

  ```
  SELECTOR executerSelectPage: "<tab name>"
  ```

  (the tab name has to be specified exactly.)

1. Upper and lower case are not taken into consideration in the search.

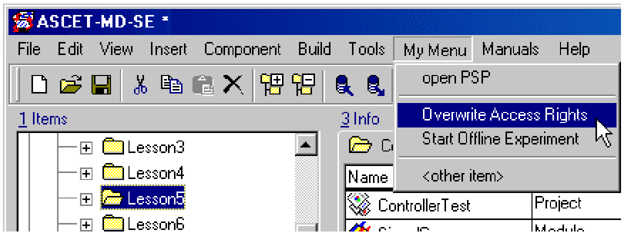
3.4.5 MENUITEM

You can use the `MENUITEM` keyword to invoke an existing menu item from the active window.

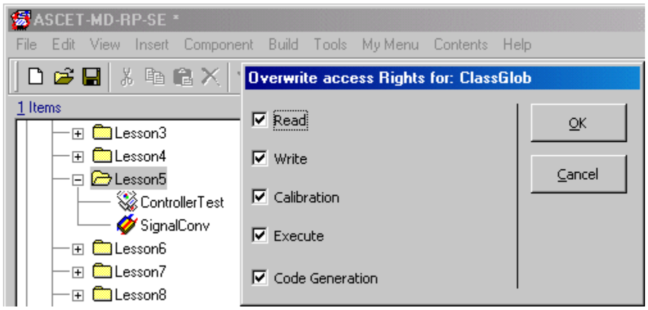
Example:

The **My Menu** menu was added in the Component Manager; the menu function **Overwrite Access Rights** invokes the `set_access.txt` file with the following contents:

```
MENUITEM Item>Change Access Rights>Overwrite
```



If this menu item is invoked, the "Overwrite Access Rights for ..." window is opened for the selected database entry.



Details on access rights for database entries can be found in section 2.3.7 „Database Access“ on page 139.

3.4.6 FILE

The keyword `FILE` is used to invoke a script file from a script file. The execution of the first script file is interrupted as long as the second one is being executed.

Example:

- `File set_access.txt` invokes the `set_access.txt` script file from the current script file.

3.4.7 FORK

The keyword `FORK` is used in exactly the same way as `FILE` to invoke a further script file. The difference between them is that the execution of the first script file is not interrupted.

3.4.8 SEND

The `SEND` keyword is used to send a (Windows) message to another running application. ASCET waits until the application has processed the message. The following possibilities to send a message are available:

```
SEND <class> <message>
SEND <class> <window> <message>
SEND <class> <window> <message> <lparam>
SEND <class> <window> <message> <lparam> <wparam>
```

`<class>` is the class name of the application.

`<window>` is the window title of the application.

`<message>` is the name or identifier of the message to be sent.

`<lparam>` is the first optional parameter (0 ... 4294967295).

`<wparam>` is the secondoptional parameter (0 ... 4294967295).

For more details, refer to the Microsoft Windows Win32 interface documentation, checking the `SendMessage` keyword in particular.

3.4.9 POST

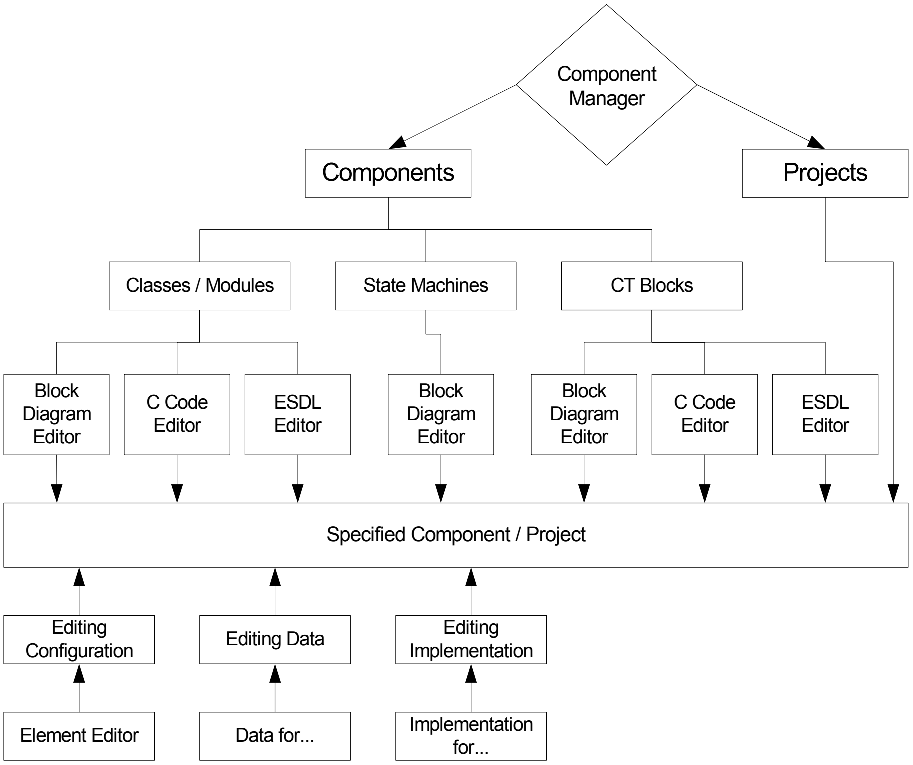
The `POST` keyword is used, like `SEND`, to send a (Windows) message to another running application. Here, however, ASCET does not wait for the application to process the message.

For more details, refer to the Microsoft Windows Win32 interface documentation, checking the `PostMessage` keyword in particular.

4 Specification of Components and Projects

The Embedded Control System is specified in a project. A project in its turn is made up of different components.

The content (declaration of variables, parameters), defined interfaces (processes, methods), and an algorithm which defines the processing, are specified in a component.



Components can be specified graphically, in ESDL or C code.

Each element used for specification has default values for the configuration, the implementation and the data. Editors can be used to modify these values.

This chapter describes working with the different specification editors and how to edit the attributes of the elements used.

4.1 The Block Diagram Editor

This section describes in detail how to use the block diagram editor. It focuses on using the program rather than developing any particular functionality. The graphic language in ASCET is described in detail in "The Modeling Language" in the ASCET reference guide.

The ASCET block diagram editor is used to specify components graphically. The user draws a block diagram that determines what the component does.

A component can contain basic and complex elements, operators, control flow statements and references to other components. Section "To create a database item:" on page 75 describes the creation of a component.

To open the block diagram editor:

- In the Component Manager, select the desired item.
- Double-click on the item

or

- select **Component** → **Edit Item**

or

- select **Edit Item** from the context menu

or

- press <ENTER>

  The selected item opens in the block diagram editor.

  When you open a component that exceeds the currently selected size of the drawing area (see "To set the size of the drawing area:" on page 181), a message window opens. It displays the size of the drawing area and the size of the component, and offers the possibility to adjust the former.

To exit the block diagram editor:

- In the block diagram editor, select **Component** → **Exit**

or

- click on the **X** at the right side of the title bar.

  If any diagram contains unsaved changes, you are asked whether you want to save the changes.

- Activate the option **Remember my Decision** if you want to give the same answer to *all* questions of this type.

  In that case, the "Save Changes in Graphic" window no longer opens. You can revoke this setting in the ASCET options window, "Confirmation Dialogs" node (see "Options for Confirmation Dialogs" on page 45).

- Click **Yes** to confirm the saving.

  The changes are stored to the cache; the block diagram editor is closed.

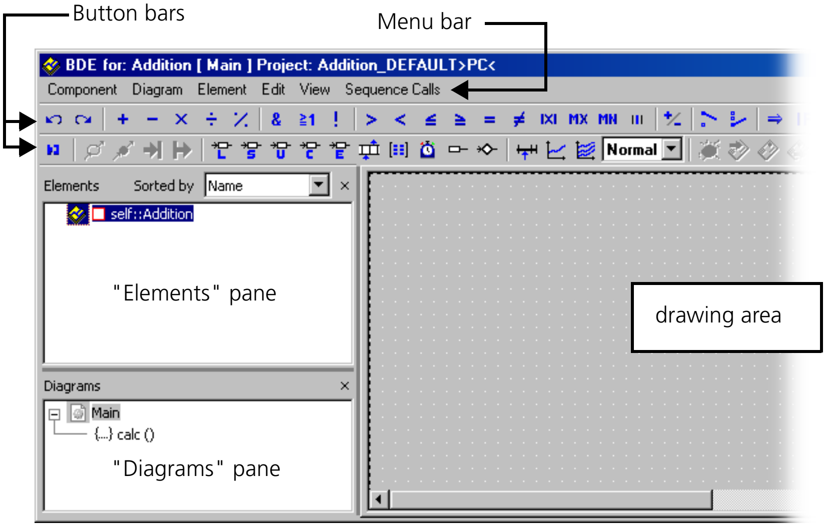
- Click **No** to reject the changes.

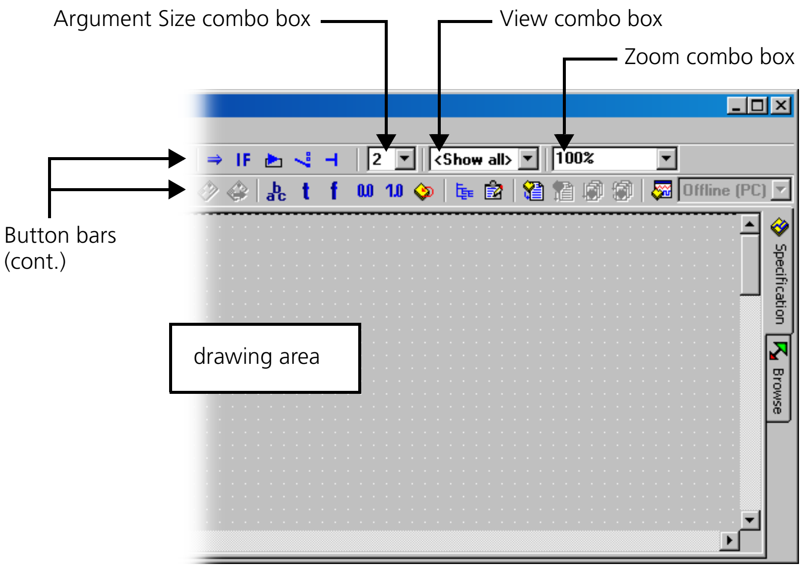
  The block diagram editor is closed without saving the changes.

- Click **Cancel** to abort closing the block diagram editor.

4.1.1 The Block Diagram Editor User Interface

The block diagram editor has two tabs with different display modes. These two modes have different functionality. Click on a tab to switch modes.





The illustrations show the block diagram editor in specification mode with the most important parts labelled.

The title bar of the block diagram editor contains the following information about the current block diagram: Component name, current diagram name, current project name and current target.

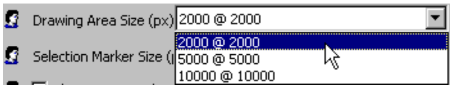
The *drawing area* is where the actual block diagram is created. The elements of the component (i.e. the inputs, outputs and variables) are shown in a tree structure in the "*Elements*" *pane*; the 'root' is the component name itself. The diagrams and methods or processes are shown in the "*Diagrams*" *pane*. The "Elements" and "Diagrams" panes can be hidden or shown, if required.

The size of the drawing area can be adjusted in the ASCET options window. Keep the following in mind: If you select a size larger than the fixed size (2000 x 2000 px) of previous ASCET versions (V5.2.0 and earlier), and use the larger area for modeling, the component cannot be completely displayed in older versions. Elements outside the drawing area cannot be deleted from the component because you cannot remove their occurrences from the drawing area.

To set the size of the drawing area:

The size you set applies to all graphical editors—block diagram editor (including CT blocks), state machine editor, project editor ("Graphics" tab). Currently open windows, however, are *not* affected by a change.

- In the Component Manager, select **Tools** → **Options** to open the ASCET options window.
- Open the "Block Diagram" node (cf. page 54).
- In the "Drawing Area Size (px)" combo box, select a size.


Drawing Area Size (px) 2000 @ 2000
Selection Marker Size (

2000 @ 2000
5000 @ 5000
10000 @ 10000

If you selected a size larger than `2000 @ 2000`, the following message is displayed:

```
If a size > 2000 @ 2000 px is
selected, components may be
displayed incompletely in
ASCET versions < V5.2.1!
```

- Confirm the message with **OK** to select the size.

- Click **OK** to close the options window.

  The next time you open the block diagram editor (or another graphical editor), the drawing area has the selected size.

To open an oversized component:

ly you open a component that exceeds the currently selected size of the drawing area, a message window opens. It displays the current size of the drawing area, as well as the component size, and it offers the possibility to resize the drawing area.

- Activate the option **Remember my Decision** if you want to give the same answer to *all* questions of this type.

  In that case, the "Save Changes in Graphic" window no longer opens. You can revoke this setting in the ASCET options window, "Confirmation Dialogs" node (see "Options for Confirmation Dialogs" on page 45).

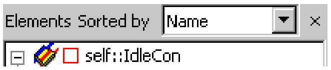
- Click **Yes** to selct a suitable size for the drawing area.

  The size is selected so that the component is displayed completely.

- Click **No** if you do not want to change the size of the drawing area.

  You cannot see the entire component. Elements outside the drawing area cannot be deleted.

To show or hide the lists:

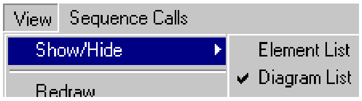


- Click on the **X** next to the name of the list you want to close

or

- select**View** → **Show/Hide** → **Element List** or **Diagram List**.

  The selected list is hidden. The respective tick mark in the **View** → **Show/Hide** menu is removed.

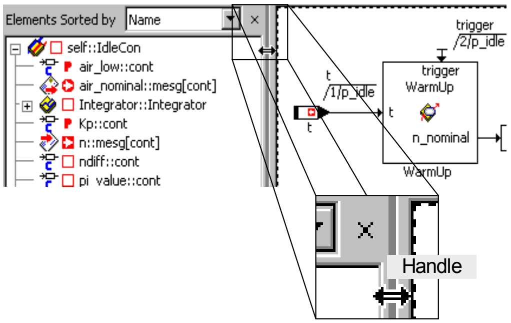


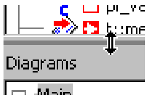
- To redisplay the list, select **View** → **Show/ Hide** → **Element List** or **Diagram List** once again.

To adjust width and height of the lists:

- Move the mouse cursor over the split bar located between the "Elements" list and the drawing area.

  The mouse cursor turns into a resize handle.

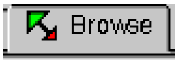




- Drag the split bar to the right or to the left to adjust the width of the lists as needed.
- Drag the split bar between the "Elements" and "Diagrams" lists up and down to adjust the height.

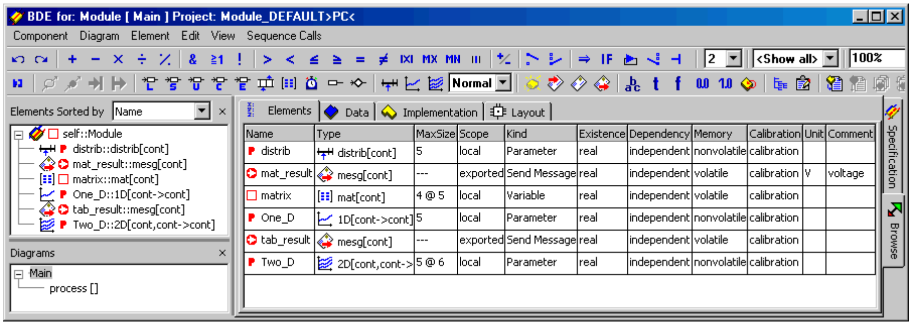
In order to gather more information about the elements of the component specified you can switch to browser mode in all component editors.

Switching display mode:



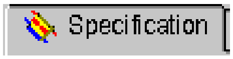
- Click on the **Browse** tab.

  The browser mode is displayed:



Name	Type	MaxSize	Scope	Kind	Existence	Dependency	Memory	Calibration	Unit	Comment
distrib	distrib[cont]	5	local	Parameter	real	independent	nonvolatile	calibration		
mat_result	mesg[cont]	---	exported	Send Message	real	independent	volatile	calibration	V	voltage
matrix	mat[cont]	4 @ 5	local	Variable	real	independent	volatile	calibration		
One_D	1D[cont->cont]	5	local	Parameter	real	independent	nonvolatile	calibration		
tab_result	mesg[cont]	---	exported	Send Message	real	independent	volatile	calibration		
Two_D	2D[cont,cont->	5 @ 6	local	Parameter	real	independent	nonvolatile	calibration		

This view offers the same functionality and structure as the view concept of the Component Manager (see chapter 2.1.3 and chapter 2.3.2). The menu functions, however, are here only available as context menus in the various tabs.

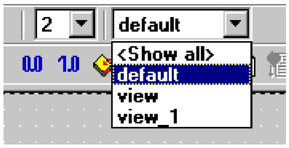


- Click on the **Specification** tab to return to the specification view.

To select a view of the diagram:

Details on views are given in chapter 7.3 "Views" on page 683.

- From the "View" combo box in the upper right corner of the toolbar, select a view.



The view changes according to the view settings of the diagram items. All items that are marked invisible for the selected view are hidden.

Description of Menu Options

- **Component**

– *Clean code generation directory*

  Deletes all files in the code generation directory.

– *Touch*

  Forced regeneration during the next code generation
  (*Flat* → the edited component,
  Recursive → the referenced components also).

– *Generate Code*

  Generates the code for a component.

– *Compile*

  Compiles the generated code.

– *Open Experiment*

  Starts an experiment.

– *Default Project*

  Allows editing the default project used for experimenting with components
  (*Edit* → opens an editor for the default project,
  Resolve Globals → automatically creates a global element for each imported element in the component for which there is no exported element,
  Delete Unused Globals → deletes unused global elements).

– *Edit Layout*

  Opens the layout editor.

– *Edit Data*

  Opens the data editor for a component. Search of component data is possible.

– *Edit Implementation*

  The implementation editor opens. Search of component implementations is possible.

– *Edit Notes*

  Opens the notes editor - you can make notes about the component here.

– *Check Dependency*

  Checks whether the allocation of formal parameters to the model parameters is correct.

- *Export Data*

  Exports a component data set.

- *Delete Unused Elements*

  Deletes the elements that do not appear in the diagram from the "Elements" pane (deactivated for state machines).

- *Show Path*

  Shows the path of an included component.

- *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard
  (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path
  Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

- *File out Generated Code*

  Saves the code generated in the file system
  (*Flat* → the edited component,
  Recursive → the referenced components also).

- *View Generated Code*

  Generates the code for the component and displays it in a text editor.
  The text editor can be selected in the ASCET options window, "ASCII Editor" node (cf. page 60).

- *File Out Generic Code For External Make*

  Saves the code generated in any directory for processing at a later point using external tools, e.g. an external Make/Build process.

- *Exit*

  Exits the block diagram editor.

• **Diagram**
(also available as context menu in the "Diagrams" pane)

- *Store to Cache*

  The component specification is saved in the cache. (It is not saved permanently in the database.)

- *Analyze Diagram*

  Analyzes the current diagram.
- *Load Diagram*

  Loads a diagram.
- *Add Diagram*

  Creates a new diagram.
  (*Public* → contains only public methods,
  Private → contains only private methods)
- *Rename Diagram*

  Renames a diagram
- *Delete Diagram*

  Deletes a diagram from the component.
- *Add Method*

  Creates a method. Available for classes and modules.
- *Add Process*

  Creates a process.
- *Add Trigger / Action / Condition*

  Only available in the state machine editor, see "Special Menu Options" on page 268.
- *Edit*

  Editing the method/process interface.
- *Copy*

  Copies the selected method/process.
- *Rename*

  Renames a method/process.
- *Delete*

  Deletes a method/process.
- *Move Up*

  Moves a diagram or a method/process (upwards).
- *Move Down*

  Moves a diagram or a method/process (downwards).
- *Move*
  Moves methods/processes between diagrams.

- *Edit Implementation*

  Specifies the implementation of a method / process.

- **Element**
  (also available as context menu in the "Elements" pane)

  Note

  *The **Add Item**, **Rename**, **Delete** and **Edit** functions are only available when ASCET-MD is installed.*

  - *View* → *Collapse all* is used to collapse the tree structure in the "Elements" pane so that only the component is shown.
  - *View* → *Expand all* expands the tree structure so that the entire content of the component is visible.
  - *Add Item*

    Accepts a component as a complex element.
  - *Rename (*<F2>*)*

    Renames a selected diagram element.
  - *Delete (*<DEL>*)*

    Deletes a selected diagram element.
  - *Show Occurrences*

    Displays all occurrences of an element (block diagram editor only).
  - *Edit*

    Edits the configuration of a selected element.
  - *Edit Data*

    Edits the data of a selected element.
  - *Edit Implementation*

    Edits the implementation of a selected element.
  - *Set Cache Locking*

    These menu options are used in connection with ASCET-RP. They are described in the ASCET-RP user's guide.
  - *Edit Component*

    Opens the specification editor for a selected included component
  - *Replace Component*

    Replaces a component with another component. The name of the old component remains.

- *Show Path*

  Displays the path of a selected included component.

- *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path
  Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

- *Notes*

  Opens the notes editor for an included component.

- *Edit Distribution*

  Opens an editor for the selected distribution (only in group tables).

- *Edit Max Size*

  Opens an editor where you can specify the maximum size of an array, any table or matrix.

- *Copy Elements (*<CTRL> + <C>*)*

  Copies one or more selected elements from the "Elements" pane.

- *Paste Elements (*<CTRL> + <V>*)*

  Pastes one or more copied elements into the "Elements" pane.

- *File Out Data*

  Writes the data from an array or a table to a file.

- *File In Data*

  Reads the data for an array or a table from a file

- *Export Data*

  Exports the data set of an included component.

• **Edit**

- *Cut (*<CTRL> + <X>*)*

  Cuts (deletes) a diagram element.

- *Copy (*<CTRL> + <C>*)*

  Copies a diagram item.

- *Paste (<*CTRL*> + <*V*>)*

  Pastes a diagram item.
- *Delete (<*DEL*>)*

  Deletes a connection or an element.
- *File In Buffer*

  Reads a diagram from a file.
- *File Out Buffer*

  Stores parts of a diagram in a file.
- *Views*

  Opens the "Views" dialog window. The views in which the diagram item can currently be seen are selected and can be edited.

- **View**
  - *Show/Hide*

    Shows/hides several parts of the editor or the component. (*Element List* → "Elements" pane,
    Diagram List → "Diagrams" pane,
    Connected Elements → browser area for elements connected to a selected diagram element (see page 221),
    Impl. Casts in Element List → implementation casts in the "Elements" pane; if nothing else has been selected for the current view, the implementation casts remain visible in the drawing area.
  - *Redraw*

    Redraws the diagram.
  - *Rebuild Connections*

    Smoothes fragmented connections that were created during automatic conversion of operator implementations (see page 508).
  - *Undo (<*CTRL*> + <*Z*>)*

    Reverses the most recent action.
  - *Redo (<*CTRL*> + <*Y*>)*

    Reverses an undo command.
  - *Show Hierarchy Path*

    Displays the complete hierarchy path for the component.
  - *Parent Component*

    Opens an editor for the including component. (The option is only available if an included component is being edited.)

  – *Parent Hierarchy Level*

    Displays the hierarchy path.

  – *Page frame Portrait*

    Displays the diagram in portrait format.

  – *Page frame Landscape*

    Displays the diagram in landscape format.

  – *Grid*

    Modifies the grid in the drawing area:

  – *Print Diagram*

    Prints the component.

  – *Save as Postscript/ Bitmap/ RTF/ GIF*

    Saves the diagram in the format specified.

- **Sequence Calls**

  – *Sequencing - Ignore Info*

    Automatic assignment of sequence calls.

  – *Sequencing Starting With*

    Automatic assignment of sequence calls starting with a particular number.

  – *Sequencing Appending*

    Appends sequence calls to an existing sequence.

  – *Scale To Step Size*

    Scales sequence calls
    (*For Diagram* → for the whole diagram,
    For Method/Process → for a single method/process)

  – *Reset*

    Resets sequence calls
    (*For Diagram* → for the whole diagram,
    For Method/Process → for a single method/process,
    For Selected Blocks → for selected blocks)

  – *Show*

    Shows groups of sequence calls
    (*For Diagram* → for the whole diagram,
    For Method/Process → for a single method/process,
    For Selected Blocks → for selected blocks,
    Unused → unused sequence calls)

- *Hide*

  Hides groups of sequence calls
  (contains the same menu functions as **Sequence Calls → Show**)
- *Next Seq. Call (<CTRL> + <CURSOR RIGHT>)*

  Moves between sequence calls (next call).
- *Previous Seq. Call (<CTRL> + <CURSOR LEFT>)*

  Moves between sequence calls (previous call).

4.1.2 Defining a Component Interface

The first step in specifying a component is to define its interface. The interface determines how the component interacts with other components, i.e. what data it receives and passes on, and how it can be addressed. The interface is the main difference when specifying different types of components in the block diagram editor; therefore it is explained separately for classes and modules.

Classes

The interface of a class consists of its public methods and their arguments and return values. The arguments form the input parameters of the class, the return values form the output parameters. The methods can be stimulated externally to trigger the calculations within a component.

To create a method:

- Select **Diagram → Add Method**.

  A new method is created in the "Diagrams" pane.
- Type in a name for the method and press <ENTER>.

Every class contains one or more methods; the default method `calc` is created automatically. A method can have a number of arguments and a return value. The arguments and the return value can be modified using the interface editor.

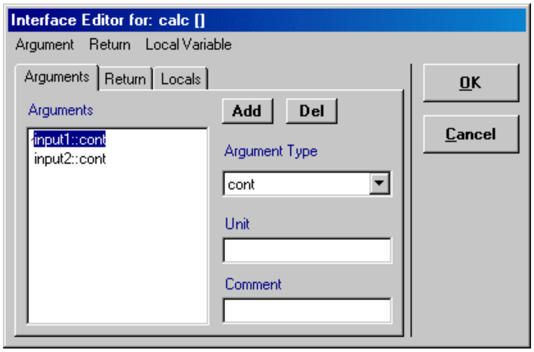
To edit the interface of a method:

- In the "Diagrams" pane, select the name of the method you want to modify.
- Select **Diagram → Edit**

or

- double-click on the method.

  The interface editor for the selected method opens. In this dialog box you can assign arguments and a return value to the method, and define local variables.



- Define the arguments, return value and local variables for the method as described below.
- Click **OK** to close the Interface Editor.

To add an argument to the method:

- In the interface editor, activate the "Arguments" tab.
- Click on the **Add** button

or

- select **Argument** → **Add**.

  The new argument is added to the "Arguments" list with its name highlighted for in-place editing.
- Name the argument and press <ENTER>.
- Specify the type of argument by selecting the value you want from the "Argument Type" combo box.

  The default argument type is `cont`.
- When you selected an array or matrix as type, select **Argument** → **Edit Max Size**.

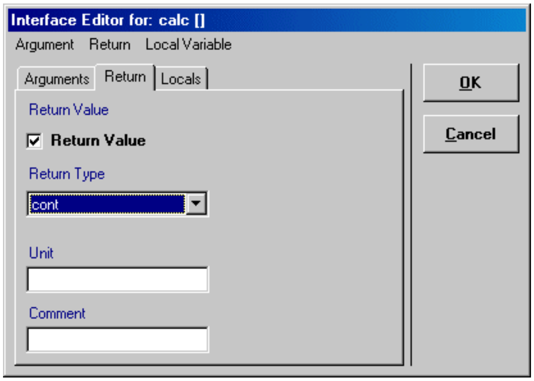
- In the "Max Size for: <argument>" window, enter the array or matrix size.
- Enter a unit for the argument in the "Unit" box.

  The unit is purely descriptive and does not influence the functionality of the class.
- Type a comment relating to the argument into the "Comment" box.

  Every argument or return value can have a comment associated with it. This comment is displayed when documentation for the component is generated.

To add a return value to the method:

- Activate the "Return" tab of the interface editor.



Return Value

☑ Return Value

Return Type

cont

Unit

Comment

- Tick the **Return Value** box, if the method is to have a return value.
- Specify the data type of the return value.
- When you selected an array or matrix as type, select **Return** → **Edit Max Size**.
- In the "Max Size for: return" window, enter the array or matrix size.
- Type in a comment and a unit.

To add local variables to the method:

- Activate the "Locals" tab of the interface editor.

  The "Locals" tab contains the same elements as the "Arguments" tab.
- Click on the **Add** button

or

- select **Local Variable** → **Add**.

  A new local variable is added to the "Local Variables" list.
- Enter a name for the local variable.
- Specify the type for the local variable.
- When you selected an array or matrix as type, select **Local Variable** → **Edit Max Size**.
- In the "Max Size for: <local>" window, enter the array or matrix size.
- Enter a comment and a unit for the variable.

To edit arguments and local variables:

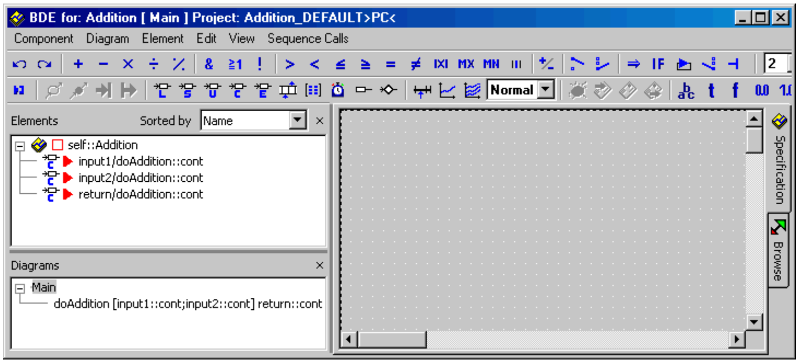
- Select **Argument** → **Rename** to rename an argument.
- Select **Argument** → **Move Up** or **Move Down** to move an argument in the list.
- Select **Argument** → **Delete**

or

- click on **Del** to delete a selected argument.

For editing local variables, the "Locals" tab offers the **Local Variable** menu, which contains the same menu functions.

The arguments, local variables and the return value are added to the "Elements" list of the block diagram editor. Each argument or return value is represented as a pin on the graphical block of the component.



Sorted by: Name

self::Addition
input1/doAddition::cont
input2/doAddition::cont
return/doAddition::cont

Main
doAddition [input1::cont;input2::cont] return::cont

To rename or delete a method:

- Select the method in the "Diagrams" pane.
- Select **Diagram** → **Rename**

or

- press <F2> to rename the method.
- Select **Diagram** → **Delete**

or

- press <DEL> to delete the selected method.

In the "Diagrams" list, the diagrams and methods are shown in chronological order by default. Every new diagram or method is added at the end of the list. The order can be modified by moving diagrams and methods within the list.

To move a diagram or method:

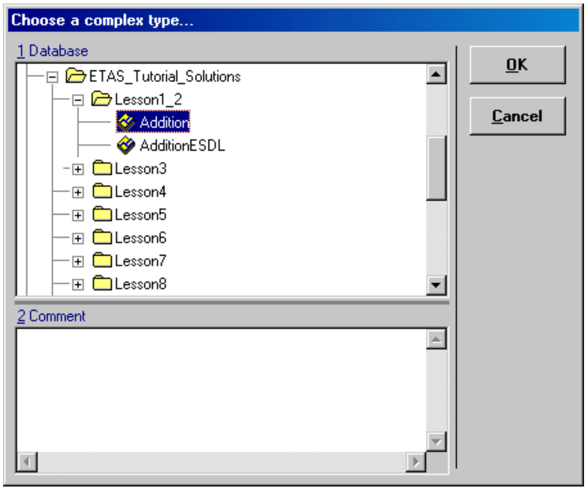
- In the "Diagrams" list, select the item you want to move.
- Select **Diagram** → **Move Up** to move the selected item one position up in the list.
- Select **Diagram** → **Move Down** to move the selected item one position down in the list.

You can use other components as interface elements of components. The procedure is described here for an argument of type *user-defined*. The same procedure can be used for a complex return value.

To assign a component as an argument:

- Open the Interface Editor for the method to which you want to add the component.
- From the "Arguments" list, select the item for which you want to specify the data type (or add a new item).
- In the "Argument Type" list, select the entry `<user defined>`.

  A selection dialog box opens for the current database.



- From the "1 Database" list, select the component you want and click **OK** to close the dialog.

  The selected component appears in the "Argument Type" pane on the interface editor.
- Click **OK** to store the changes you made to the interface.

Modules

The interface of a module consists of processes. A default process is created automatically for every new module. Processes determine the activation of the module functionality, but they do not define inputs or outputs. Modules communicate and interact using messages and global elements (imported, exported elements).

To create a process:

- Select **Diagram** → **Add Process**.

  A new process is created and its name is highlighted to allow in-place editing in the "Diagrams" pane.
- Enter a name for the process and press <Enter>.

  You can move (cf. page 196), rename or delete (cf. page 196) processes the same way as methods.

As a process does not have any arguments, only the "Locals" tab and the **Local Variable** menu (see page 195) are available in the interface editor.

4.1.3 Complex Types as Interface Elements

This section explains, with the help of examples, the use of composite and user-defined elements as interface elements.

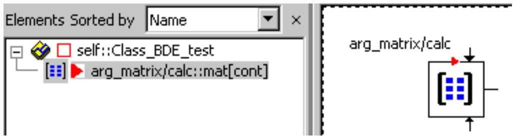
Arrays and Matrices

When using a composite or user-defined interface element (array, matrix, class), you can access it via the normal element pins. In addition, you can use the Get and Set ports to access the entire data structure. A matrix is used as an example to describe the procedure.

To create the matrix argument:

- Add an argument with argument type `mat(cont)` to a method (cf. page 193).

- Place the argument in the drawing area.



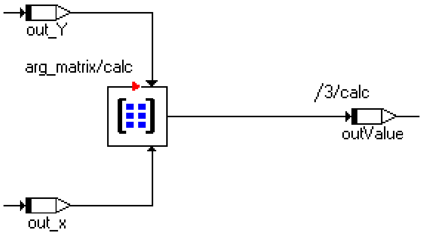
Elements Sorted by Name

self::Class_BDE_test

arg_matrix/calc::mat[cont]

The element has only the readout pins (see also "Arrays and Matrices" in the ASCET reference guide), it is not possible to write to the matrix.

- Use the normal readout pins, as shown in the figure.

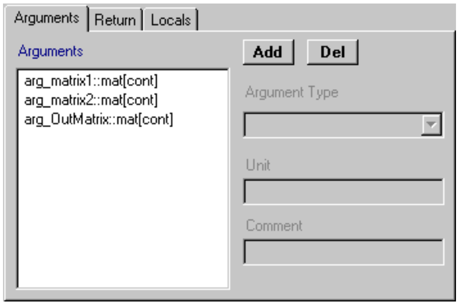


The procedure to return matrices or arrays is more complicated. It is described in the following paragraphs, using the addition of two matrices as an example.

To create the computation class:

- Create a class with a method to contain the matrix calculation.
- In the method, add two arguments of type `mat(cont)` for the matrices to be added.

- Add another argument (e.g., `arg_OutMatrix`) of the same type for the result matrix.



Arguments

```
arg_matrix1::mat[cont]
arg_matrix2::mat[cont]
arg_OutMatrix::mat[cont]
```

Since you cannot write to the argument, you need an auxiliary matrix.

- Use the **Matrix** button to create a matrix of type `cont`.
- In the "Max Size for" window, accept the preset values.

  The third argument is assigned to the auxiliary matrix via the Get port. Thus, the size of the latter is determined, and does not have to be explicitly entered.

Note

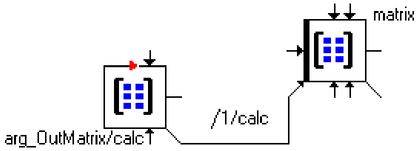
*Since you assigned no special size to the matrix arguments, the preset size **must be** accepted. If not, the code generation returns an error message.*

To specify the matrix addition:

- Place the auxiliary matrix and the argument `arg_OutMatrix` in the drawing area.
- Right-click on each element, and select **Get/Set Ports** from the context menu.

  The Get port of the argument and the Get and Set ports of the auxiliary matrix are displayed in the diagram.

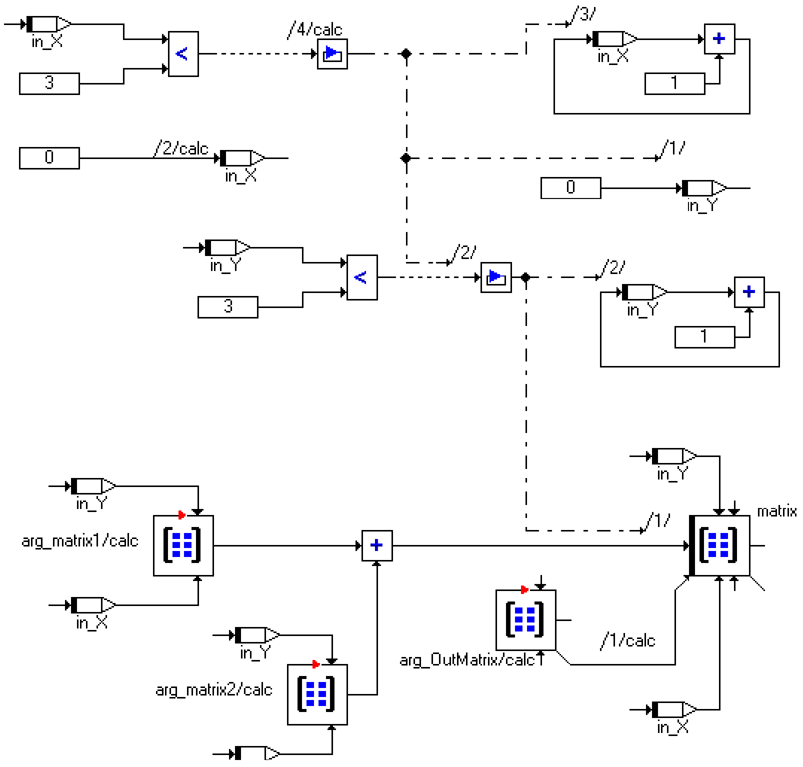
- Assign the argument `arg_OutMatrix` to the auxiliary matrix via its Set port.



- Select the sequence number 1 for the sequence call, so that the assignment is executed as the first step of the method. (Sequence calls are described in detail in section "Sequence Calls" on page 234).

  The auxiliary matrix accesses the memory area of `arg_OutMatrix`. Thus, the calculation results are made available outside the method.

- Create the necessary indices, and specify the matrix addition.



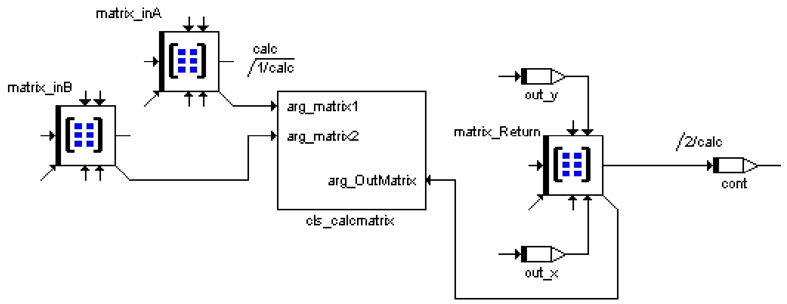
The example uses a 3x3 matrix. The loops (but *not* the sizes of the matrix arguments and auxiliary matrix!) have to be adapted to the actual conditions.

To perform the calculation, you have to create another class or module that assigns real values to the arguments of the computation class.

To perform the calculation:

- Create a class or module.
- Use **Element** → **Add Item** to include the computation class as a complex element (cf. page 217).

- Create two matrices of type `cont`, and fill them with the input data.
- Create a third matrix of the same type and size for the result.
- Place the elements in the drawing area, and connect them as shown below.



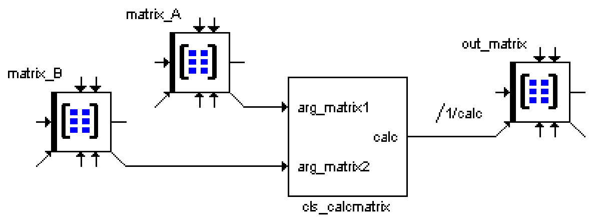
- Use the output pins of the result matrix to read it.

Note

When you assigned a size to these matrices other than the preset values in the "Max. size" window, the code generation will produce the following ***warning****:*

```
type mismatch in array max size: <1@1> and <x@y> -
accepted since larger matches smaller
```

A frequently tried approach to return a matrix or an array is to add a return value of the respective type to a method, and access the return value via the Get/Set ports. The following figure shows such an arrangement; the class `calc_matrix` contains the `calc` method with two matrix arguments and a matrix return value.



This does not work! The ports transfer pointers; in this case, pointers to a structure within the method are. This structure, though, is only available while the method is computing - which means that, in the example, the data no longer exist at the time of the assignment to `out_matrix`.

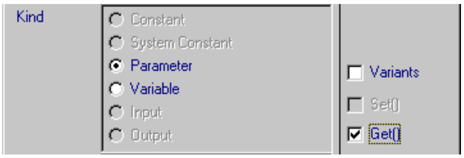
Characteristic Lines and Maps

You cannot directly pass a characteristic line or map as argument. To do so, you have to include the characteristic table in a class, and use the class as an argument. The procedure is described for a characteristic line.

Preparations:

- Create a class, e.g., `Class_charline`.
- In the `Class_charline` class,create and set up a characteristic line via the **One D Table Parameter** button (cf. page 211).

  You have to make sure that the table can be accessed from outside the class. For that purpose, proceed as follows.
- Open the element editor for the characteristic line (see also "Element Configuration" on page 448).
- Add an output for the characteristic line by activating the **Get()** option.

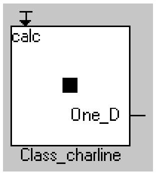


Kind
- Constant
- System Constant
- [✓ Parameter]
- Variable
- Input
- Output

- [ ] Variants
- [ ] Set()
- [✓] Get()

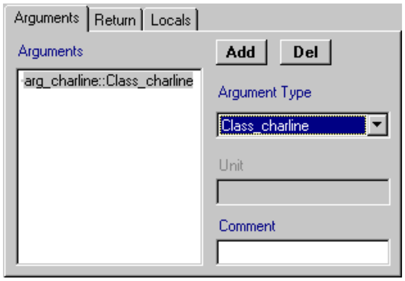
- Close the element editor.

  An output for the characteristic line is added to the class layout.



To pass the characteristic line as method argument:

- Create a class that is to contain the characteristic line as method argument.
- Add the class `Class_charline` as an argument of type `<user-defined>` to one of the methods, as described on page 197.



arg_charline::Class_charline

Argument Type: Class_charline

- Click **OK** to close the interface editor.

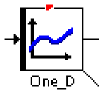
To use the characteristic line as method argument:

To actually use the method argument, proceed as follows.

- Place the complex argument in the drawing area.

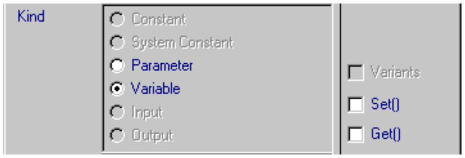
  Unlike with matrix and array arguments, you cannot access the pins of the characteristic line. Therefore, you have to pass it to a local characteristic line.

- Create and set up a characteristic line (cf. page 211), and place it in the drawing area.
- Right-click on the characteristic line, and select **Get/Set Ports** from the context menu.



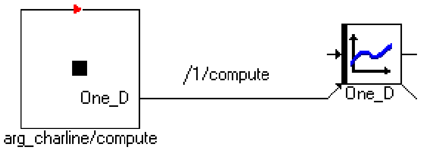
  Characteristic tables are created as parameters, i.e., they cannot be written from within the program. Thus, only the Get port is made available.

- Open the element editor for the characteristic table, and assign the kind **Variable** (see also "Element Configuration" on page 448).


Kind
( ) Constant
( ) System Constant
( ) Parameter
(●) Variable
( ) Input
( ) Output
☐ Variants
☐ Set()
☐ Get()

Now the Set port is available, too, and you can assign the complex argument to the characteristic line.

- Connect the argument output to the Set port of the characteristic line.



- Select the sequence number 1 for the sequence call, so that the assignment is executed as the first step of the method.

Note

If this assignment is not performed as the first step of the method, inconsistencies arise.

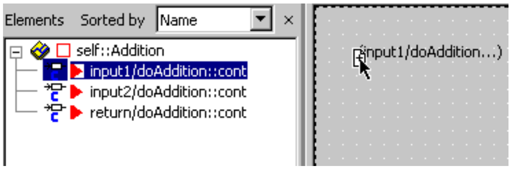
You can now analyze the characteristic line as usual. Data are read from the memory area in which the characteristic line passed as argument is located.

4.1.4 Creating Block Diagrams

After the interface for a component has been defined, the arguments and return values or the inputs and outputs are shown in the "Elements" pane of the block diagram editor. They can now be arranged on the drawing area and connected to other graphical elements, such as variables or operators.

To place an interface element:

- From the "Elements" pane, select the element you want and drag it to where you want it in the drawing area.



The element is positioned in the drawing area.

- You can select and drag the element in the drawing area to move it to another position.

To create a basic element:

- Click on the button for the element you want to create in order to load the mouse cursor with the corresponding type of element.

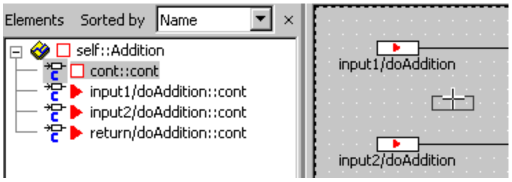
  The element buttons are in the second row of the button bar.

  The Element Editor of the element is opened automatically and allows the user to change the element's properties immediately.

Note

If it is not desired to open the Element Editor upon an element's creation, deactivate the **Always show Edit Dialog for new elements option** *at the bottom of the editor or the* **Edit primitive Element** *option in the in the "Confirmation Dialogs" node (page 45) of the ASCET options window.*

- Click inside the drawing area to place the new item.



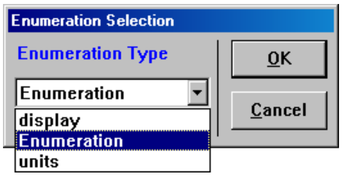
In the diagram, the element appears at the point where you clicked. You can drag it to another position.
In the "Elements" list, a new element is added.

You can change the properties of an element later by right-clicking on it. This opens a context menu with various commands. These commands are explained in section "Editing Element Properties" on page 447.

To insert an enumeration:

- Click on the **Enumeration** button.

  The "Enumeration Selection" window opens.



All enumerations defined in the database are displayed in the "Enumeration Type" combo box.

- Select the enumeration you want from the combo box.
- Click **OK** to close the selection window.

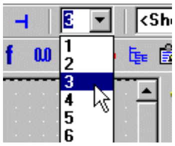
  This loads the mouse cursor with the enumeration type. The element editor is opened

- Adjust the element’s properties according to your needs and click **OK**.

- Click in the drawing area.

  This positions the enumeration.
- Enter a name for the enumeration and press <ENTER>.

To position an operator:



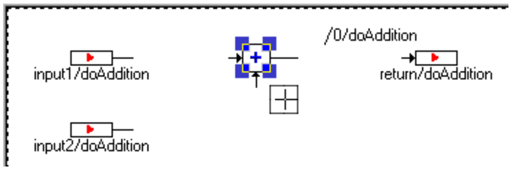
- From the "Argument Size" combo box, select the number of inputs for the operator.

Note

For some operators (e.g. division, subtraction, comparison operators) the number of inputs cannot be changed. In these cases, the selection has no effect.

- From the toolbar, select the operator you want to create. This loads the mouse cursor with that operator.
- Click inside the drawing area to position the operator.

  The operator is added to the diagram. You can adjust its position by dragging it to another position.



The flow of information in diagrams is determined by connecting the items in the drawing area.

To connect diagram elements:

- Right-click inside the drawing area, but not on any of the elements,

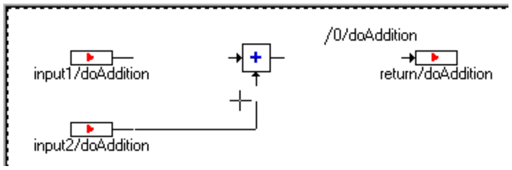
or

- click on the **Connect** button.

  This starts the connection mode, which is shown by the cursor changing into a cross hair.

- Click on the pin of the first diagram element you want to connect to start a connection.
- Move the cursor to the end point of the connection to create a connection.

  A line follows the path of your cursor movement. Whenever you click inside the drawing area but not on an element, you create an anchor point for the connecting line. Right-clicking while dragging a connection line deletes the most recent anchor point.



- Click on a pin of the diagram item that you want to connect to complete the connection.

  The connection is established and a line is drawn.

The appearance of the line depends on the connected elements.

Color/Style	Explanation
black/solid	connection between two numerical pins
black/dashed	connection between two logical pins
black/dash-dotted	control flow connection (cf. page 214)
colored (default: green)	Comment line; the sequencing for this statement or operation is still unresolved (cf. chapter 4.1.6). The color of commet lines can be selected in the ASCET options window (cf. ""Colors" node" on page 55).
red	wrong connection, e.g., between numerical and logical pins.

When you drag an element, the connection line follows. It may however be necessary to change the path of a line to keep the diagram neat and clear. You can do this by simply dragging the line.

To end the connection mode:

The connection mode is ended when a connection is complete. You can, however, end it at any time. Proceed as follows.

- Right-click in the drawing area (but not on an element)

or

- click again on the **Connect** button.

  The connection mode ends.

Arrays, Matrices, Characteristic Curves and Maps

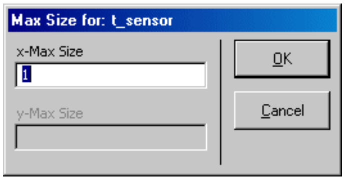
This section describes how to create arrays, matrices, characteristic lines and maps. Chapter 4.11 "Editing Data"describes how to edit the table data.

To create an array or matrix:

- Click on the **Array** or **Matrix** button.

  The element editor is opened. Since arrays and matrices are created as variables, the **volatile** attribute is set.
- Adjust the element’s properties according to your needs and click **OK**.

  The "Max size" window opens.


Max Size for: t_sensor
x-Max Size
1
y-Max Size

- Adjust the maximum number of elements in the "x-Max Size" and "y-Max Size" fields.

  The "y-Max Size" is only available when you created a matrix.
- Click **OK**.

  If you exceeded the maximum size, the following warning opens:

  `*-Max size exceeded. Reduce to limit <limit>.`

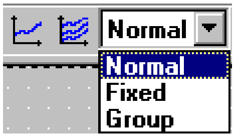
– Click **OK** to accept the limit,

or

– click **Cancel** to return to the "Max size" window.

- Place the element in the drawing area.

To create a normal or fixed characteristic line/map:



- In the button bar combo box, select the table type `Normal` or `Fixed`.
- Click on the **One D Table Parameter** or **Two D Table Parameter** button.

  The element editor is opened. Since all characteristic lines/maps are created as parameters, the **non-volatile** attribute is set and the "Memory" field is deactivated.
- Adjust the element's properties according to your needs and click **OK**.

  The "Max size" dialog box opens.
- Adjust the maximum number of sample points in the "x-Max Size" and "y-Max Size" fields.

Note

Fixed characteristic lines/maps must have at least 2 sample points per axis. Otherwise, the monotony check will produce an error.

The "y-Max Size" field is only available when you created a haracteristic map.

- Click **OK**.

  If you exceeded the maximum size, the following warning opens:

  ```
  *-Max size exceeded. Reduce to
  limit <limit>.
  ```

  – Click **OK** to accept the limit,

  or

  – click **Cancel** to return to the "Max size" window.
- Place the element in the drawing area.

To create a distribution:

Distributions are required when you are using group characteristic lines or maps (cf. "Group Tables" on page 465).

- Click on the **Distribution** button.

  The element editor opens. Since distributions are created as parameters, the **non-volatile** attribute is set and the "Memory" field is deactivated.
- Adjust the element properties to your needs.
- Click **OK**.

  The "Max size" dialog window opens.
- Adjust the maximum number of sample points in the "x-Max Size" fields.

  The "y-Max Size" is not available.
- Click **OK**.

  If you exceeded the maximum size, the following warning opens:

  ```
  x-Max size exceeded. Reduce to
  limit 2048.
  ```

  – Click **OK** to accept the limit,

  or

  – click **Cancel** to return to the "Max size" window.
- Place the element in the drawing area.

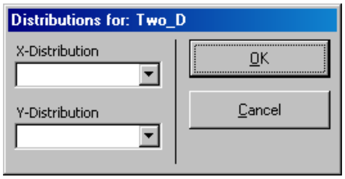
To create a group characteristic line/map:

- In the button bar combo box, select the table type `Group`.
- Click on the **One D Table Parameter** or **Two D Table Parameter** button.

  The element editor is opened. Since all characteristic lines/maps are created as parameters, the **non-volatile** attribute is set and the "Memory" field is deactivated.

- Adjust the element properties to your needs and click **OK**.

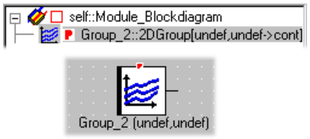
  The "Distribution for" dialog box opens.



- From the "x-Distribution" and "y-Distribution" combo boxes, select suitable distributions.

  The "y-Distribution" combo box is only available for group characteristic maps.

- Click **OK**.

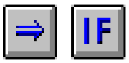

self::Module_Blockdiagram
Group_2::2DGroup[undef,undef->cont]
Group_2 (undef,undef)

  When you click on **Cancel**, no distribution is selected. The table is created nonetheless, but it is labeled `undef`.
  You have to select the distribution(s) via **Element** → **Edit Distribution** prior to using the table.

- Place the element in the drawing area.

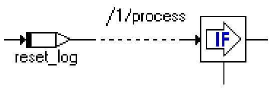
Control Flow Statements

To use the If statements:



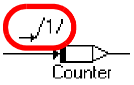
- Click on the **If then** or **If then else** button to load the mouse cursor with the respective block.
- Place the block in the drawing area.

  The If…Then block and the If…Then…Else block look nearly alike, except that If…Then…Else has two control flow branches.



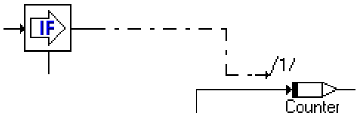
- Connect the input to a logical element.
- Specify the actions for the branches.

- Right-click on the sequence call you want to connect to a branch, and select **Connector** from the context menu.



  The sequence call becomes a connector. The colored line of an unresolved sequencing changes to black.

- Connect the desired branch to the connector.



  You can connect a branch to more than one actions. In that case, edit the connector numbers (according to page 236); as for sequence calls, each number must be unique.

- Repeat these actions for the second branch of the If…Then…Else block.

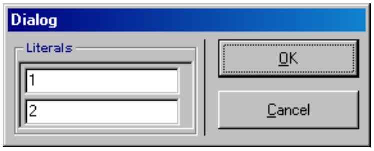
To use the switch:

- From the "Argument Size" combo box, select the number of branches for the switch.

  You must select at least two branches. The default branch is *not* counted.

- Click on the **Switch** button to load the mouse cursor with the respective block.
- Place the block in the drawing area.
- To change the values for the alternative branches, right-click on the block and select **Edit Literals** from the context menu.

  An editor window opens; it contains one input field for each branch.



- In the "Literals" field, enter the values for the branches.
- Click **OK** to accept the changes.

- Connect the input at the top of the block to an `sdisc` or `udisc` element.

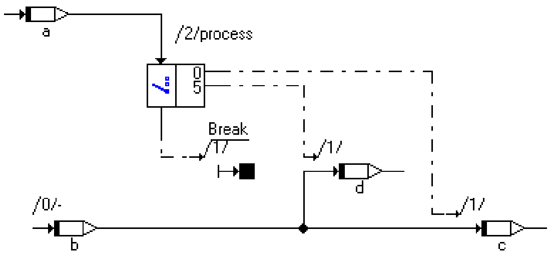
Note

If you connect the input to a `cont` *pin, an error is displayed during code generation.*

- Specify the actions for the branches.
- Right-click on the sequence call you want to connect to a branch, and select **Connector** from the context menu.

  The sequence call becomes a connector.
- Connect the desired branch to the connector.

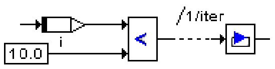
  You can connect a branch to more than one actions. In that case, edit the connector numbers (according to page 236); as for sequence calls, each number must be unique.
- Repeat these actions for the other branches.



To use the While loop:

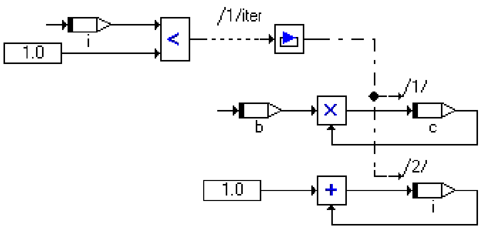
The only loop construct available in block diagrams is the `While` loop.

- Click on the **While** button to load the mouse cursor with the respective block.
- Place the block in the drawing area.
- Specify the loop condition.
- Connect the condition to the loop input.



- Specify the loop action.
- Right-click on the sequence call you want to connect to the loop output, and select **Connector** from the context menu.

  The sequence call becomes a connector.
- Connect the loop output to the connector.



You can connect the output to more than one actions. In that case, edit the connector numbers (according to page 236); as for sequence calls, each number must be unique.

Note

Make sure that you avoid infinite loops or loops unsuitable for real-time applications.

Components as Complex Elements

Components can be used as complex elements within other components. Whereas basic elements are defined in the component they are contained in, complex elements are included by reference, i.e. if the included component is changed, those changes are effective within the including component. You can connect other diagram items to the inputs and outputs of the included component.

To include a component as a complex element:

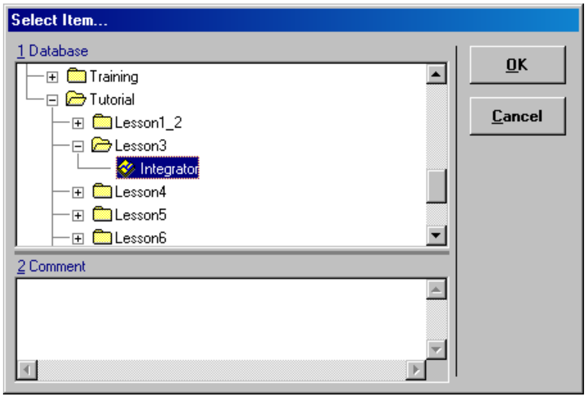
To include a component, proceed as follows.

- Select **Element** → **Add Item**

or

- Select **Add Item** from the context menu.

  The "Select Item" dialog window opens. It shows the content of the current database.



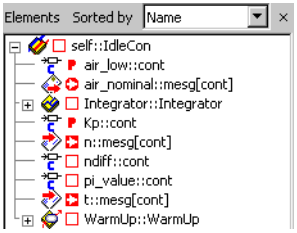
- From the "1 Database" list, select the component you want to add.
- Click **OK** to add the component.

  The component is added to the "Elements" list with its instance name highlighted to allow in-place editing.
- Edit the instance name and press <ENTER>.
- Drag the component to the drawing area to add it to the diagram.
- Connect the pins of the component in the same way as the pins of other diagram items.

Note

As an alternative to adding database items using the menu options described here, you can drag items from the Component Manager onto the block diagram editor (drag and drop).

When you include a component, its instance name is shown in the "Elements" pane. The grey triangle in front of the instance name (├ ⊞) indicates that the tree structure can be further expanded.


Elements Sorted by Name

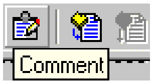
```
⊟ self::IdleCon
   ├ air_low::cont
   ├ air_nominal::mesg[cont]
   ⊞ Integrator::Integrator
   ├ Kp::cont
   ├ n::mesg[cont]
   ├ ndiff::cont
   ├ pi_value::cont
   ├ t::mesg[cont]
   ⊞ WarmUp::WarmUp
```

If you include the same component again, a new instance of the component is created and assigned a unique instance name. In the diagram, the instance name is displayed below the graphical block. In the "Elements" list, both the instance and class name are displayed, using the following format: `<instanceName>::<className>`. You can adjust the width of the list to view its entire contents.

Comments and Notes

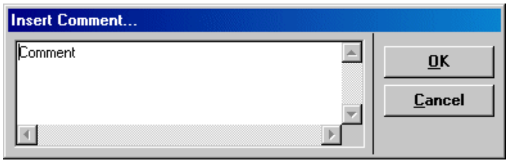
Comments do not in any way influence the functionality of a component. They only contain explanatory text that can help document your software model.

To add a comment:



- Click on the **Comment** button.

  A text box appears.


Insert Comment...

Comment

- Enter the text for your comment.
- Click on **OK**.

  This loads the mouse cursor with the comment.
- Click in the drawing area to place the comment.

You can adjust its position by dragging it to another position. You can edit the text of the comment by right-clicking on it and selecting **Edit Comment** from the context menu.

You can also attach notes to the entire component, or to an included component. The notes for a database item are entered in a separate editor window. When documentation is generated automatically, the notes are included.

To edit the notes for a component:

- If you want to change the notes of the edited component, select **Component** → **Notes**.

Or

- In the "Elements" pane or in the drawing area, highlight the included component whose notes you want to edit.
- From the context menu of the highlighted component, select **Notes**.

  The notes editor opens for the selected database item. For details, see "Notes" on page 693.

4.1.5 Editing Block Diagrams

This section describes the general editing features of the drawing area, and the features that modify the appearance of diagram items.

In some cases it is useful to retrace actions taken or to return to an earlier version of the diagram in order to rework it from there. To do this, the actions carried out in the diagram are recorded step by step on the internal clipboard. However, you can only return to an earlier version of the diagram if you are working in an open editor. When you exit from the editor, only the most recently edited version is saved; the versions recorded in the clipboard are deleted.

To undo the most recent action:

- Select **View** → **Undo**

or

- click on the **Undo** button

or

- press <CTRL> + <Z>.

  The most recent action is reversed.

Note

If you work on an earlier version of the diagram, all the later actions are deleted and the new actions are recorded.

To reverse an undo command:

- Select **View** → **Redo**

or

- Click on the **Redo** button

or

- press <CTRL> + <Y>.

  The action reversed most recently is carried out again.

Viewing Elements

To view all occurrences of an element:

- Select the element, either in the drawing area or in the "Elements" pane.
- Select **Element** → **Show Occurences**

or

- select **Show Occurrences** from the context menu

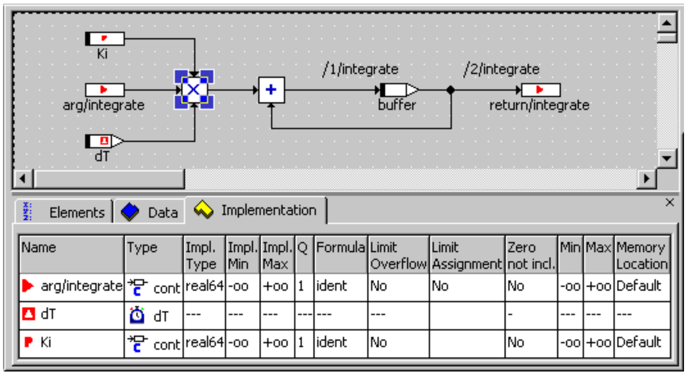
  All occurrences of the element in the current program are highlighted.

To view all elements connected to an item:

You do not have to change to the browser view if you want to look at or edit the implementations of the elements connected directly to a graphic object (element, operator, connection). This function is of particular interest for testing the implementations of the inputs and outputs of an operator which is why this is the only case described below.

- Right-click on a graphic object and select **Browse Connected Elements** from the context menu.

  A field opens containing the tabs "Elements", "Data" and "Implementation". This field lists all the elements connected directly to this graphic object.
  If the graphic object is an element, it is displayed too. Operators are not contained in the list.

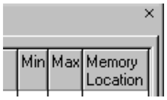


Name	Type	Impl. Type	Impl. Min	Impl. Max	Q	Formula	Limit Overflow	Limit Assignment	Zero not incl.	Min	Max	Memory Location
arg/integrate	cont	real64	-oo	+oo	1	ident	No	No	No	-oo	+oo	Default
dT	dT	---	---	---	---	---	---		-	---	---	---
Ki	cont	real64	-oo	+oo	1	ident	No		No	-oo	+oo	Default

The tabs have the same function and structure as those of the Component Manager (see chapter 2.1.3). The "Implementation" tab is activated by default.

- Work in the tabs as described in chapter 2.3.2.

  The relevant editors to edit the element properties can be opened with a double-click.



- Click the **X** at the top right of the field

or

- select **View** → **Show/ Hide** → **Connected Elements** to hide the field.
- You can redisplay the field using **View** → **Show/ Hide** → **Connected Elements** as well as with the context menu although the content is not adapted to the current selection in the drawing area in this case.

To rename or delete a diagram item:

- Select **Element** → **Rename** to rename the element.
- Select **Element** → **Delete** to delete the element.

  Before deleting an element, you must remove its occurrences from the diagram. You can use the **Show Occurences** command to select the occurrences in the diagram for removal.
- Select**Component** → **Delete Unused Elements** to delete all the elements which are listed in the "Elements" pane, but which do not appear in the diagram.

To replace a diagram item:

- Select an operator from the toolbar to load the mouse cursor with a new diagram item.
- Click on an existing operator in the diagram to replace it with the new operator.

  A confirmation prompter dialog box is displayed.
- Click **Yes** to replace the diagram item.

You can replace an element using the same procedure. In that case, you have to drag the element onto the diagram item you want to replace. Also, you can replace an operator in the diagram with an element and vice versa.

To cut, copy and paste a diagram item:

- Click on the diagram item you want to cut or copy.

  A selected item is marked by handles at its edges.
- Select **Edit** → **Cut**

or

- press <CTRL> + <X> to delete the selected diagram item and move it to the clipboard.

- Select **Edit** → **Copy**

or

- press <CTRL> + <C> to copy the selected diagram item without deleting it from the drawing area.
- Select **Edit** → **Paste**

or

- press <CTRL> + <V> to add the diagram item close to its previous location.
- Drag the new diagram item to its location.

ASCET uses its own internal clipboard for graphical information, so these operations will have no effect on the Windows clipboard. You can also copy diagram items between components, but only graphical information is copied and interface elements are excluded.

To delete a connection or an element:

- Select the connection or element in the drawing area.
- Select **Edit** → **Delete**

or

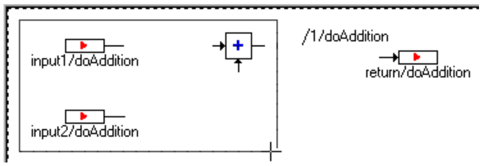
- press <DEL> to delete the selected connection or item.

To select more than one diagram item:

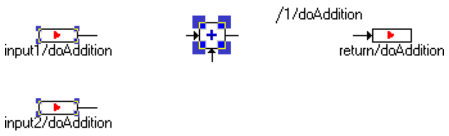
- Click on all the diagram items you want to select while holding down the <CTRL> key.

Or

- Drag a rectangle around the items you want to select in the drawing area.



The diagram items inside the rectangle are selected which is shown by the resize handles displayed along their edges.



- Cut, copy, paste, delete or move the group just as you would a single diagram item.

Appearance of Diagram Elements

To change the appearance of a diagram item:

These commands are only available in the context menu of an included component which was selected in the drawing area. They only affect the appearance of the respective occurrence.

Note

Different from the settings described in "Layout of Included Components", the settings described here cannot be included in the default layout.

- Select **Show/Hide Name** to show or hide the instance names of the occurrence.
- Select **Ports** → **Get/Set** to show or hide the get and set pins of the occurrence.

  This command is, under the name of **Get/ Set Ports**, also available for arrays, matrices, and characteristic lines/maps.

To edit the views of a diagram item:

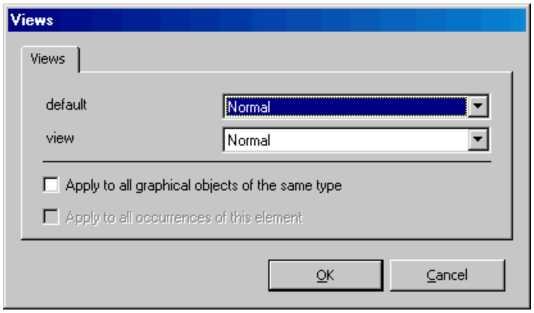
You can edit the display of individual diagram elements (basic elements, included components, operators, connections, etc.) in the different views (see chapter 7.3).

- Select the element whose display you want to edit.
- Select **Edit** → **Views**

or

- right-click the diagram element and select **Views** from the context menu.

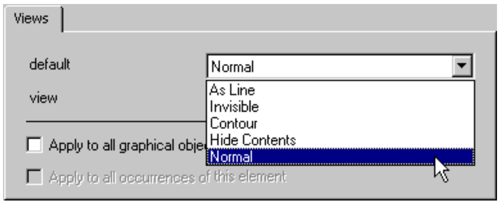
  The "Views" dialog window opens. It shows all views defined in the current database.



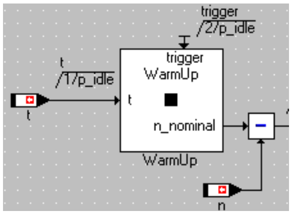
default: Normal
view: Normal

☐ Apply to all graphical objects of the same type
☐ Apply to all occurrences of this element

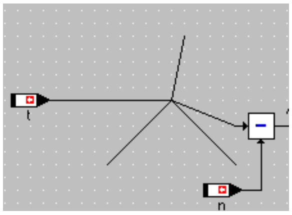
- Open the combo box of the required view.


Views
default
view
Normal
As Line
Invisible
Contour
Hide Contents
Normal
Apply to all graphical obje
Apply to all occurrences of this element

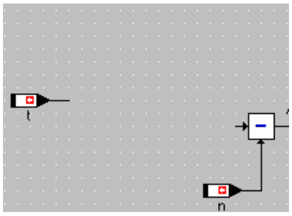
- Select the required representation for the diagram element from the combo box.



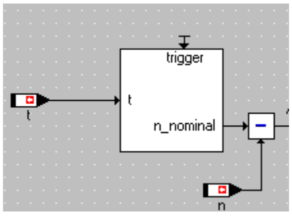
  – `Normal` - the default setting in which all available elements, such as symbols, names, sequence calls etc., are displayed.



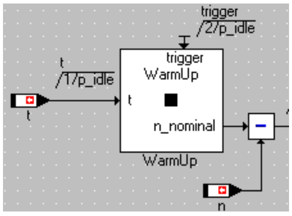
  – `As Line` - the element is replaced by one or more lines from the middle to the pins.



  – `Invisible` - the element disappears from the graphic display and can no longer be selected in the drawing area. It is, however, still in the "Elements" list.



  – `Contour` - only the contours of the element and possibly the names of the pins are shown. Sequence calls, name, symbol etc. disappear.



  – `Hide Content` - the screen display is the same as `Normal` but the component or hierarchy can no longer be opened from the drawing area.

    The content of a hierarchy thus marked is left out of the documentation, the content of a component is included in the documentation.

- Activate the **Apply to all graphical objects of the same type** option if the selection shall apply to all diagram elements of this type.
- Activate the **Apply to all occurrences of this element** option if the selection shall apply to all occurrences of this element.
- Click **OK** to confirm your selection.

  The element is displayed in the selected view in accordance with your selection.

  The command is available for all diagram elements. Views are explained in section 7.3 "Views" on page 683.

In the *As Line* mode it is not always clear that this is an element. This is why the following warning is displayed if you select an element in *As Line* mode or select a connection to this kind of element:

```
CAUTION: Element shown in "As Line" view could be
affected!
```

Layout of Included Components

If you add a component to your component (see page 217) and store it in the drawing area, the default layout of the added component, defined in the layout editor, is displayed. If this default layout does not suit your purposes, you can react in two different ways.

One possibility is to adapt the default layout in the layout editor (see chapter 4.13). Changes in the layout editor do not, however, have any influence on individual occurrences of the component in block diagrams. Existing diagrams remain unchanged; you have to replace the occurrences manually to load the changed layout.

The other possibility is to adapt the layout of a specific occurrence in the block diagram. This feature is activated via the **Activate flexible layout** option in the ASCET options window (see section "Options for Block Diagrams" on page 54). Only the layout of the edited occurrence is modified; neither the default layout nor the layout of other occurrences of the same component in the diagram is changed automatically. This means that, if necessary, you can assign each occurrence of the same component a different layout.

You can change the size of a block and move, show or hide the ports. The following must, however, be taken into consideration:

- An occurrence must at least be the size of an addition operator with two inputs.

- The minimum size of an occurrence is also limited by the number of visible ports: two ports cannot have the same position.
- If the default layout contains an icon, it is cut off if the size of the occurrence is smaller than the icon itself.

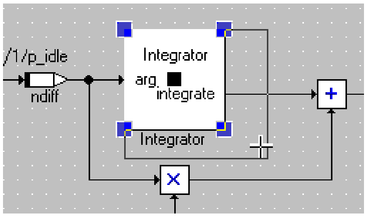
If you copy or cut out an occurrence which has been edited in this way and insert it at a different location as described on page 223, the inserted occurrence is assigned the layout of the copied/cut out occurrence.

To edit the size of an occurrence:

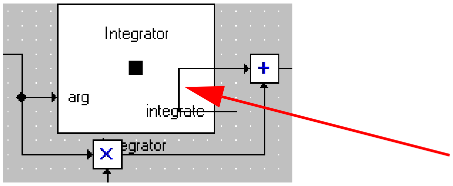
You can either edit the size of an occurrence manually or automatically set the minimum size.

- In the drawing area, mark the occurrence of the added component you want to edit.

  Four handles appear at the corners.
- Drag the handles to set the size you require for the occurrence.



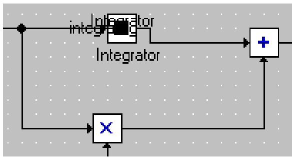
Other elements are not affected by this. Existing connecting lines remain as they are but could now look confusing.



- If necessary, rearrange the diagram elements so that everything is kept clear.

- Right-click on the occurrence and select **Minimal Size** from the shortcut menu.

  The size of the occurrence is minimized in accordance with the number of ports. The names of the ports or the occurrence are ignored.

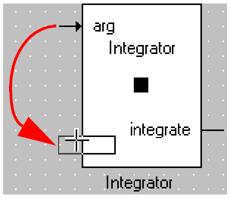


- Select **Show/Hide Name** from the context menu of the occurrence to show or hide the name of the component.

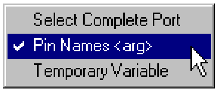
To edit ports:

You can move the ports of an occurrence and show/hide their names.

- Use the mouse to drag the port you want to move to the required position.



  You can position the port wherever there is no other port.



- Right-click on the port and select **Pin Names *<name>*** from the context menu to show/hide the display of the port name.

  If the port name is displayed, **Pin Names *<pin name>*** has a tick beside it.

- Select **Show Pin Names** from the context menu of the occurrence to show the names of all ports of the occurrence.
- Select **Hide Pin Names** to hide the names of all ports of the occurrence.

- Select **Unconnected Ports** to show or hide all unconnected ports of the occurrence. The ports of a method can only be shown or hidden together.

Note

This way, ports are only hidden, not removed. Operations like **Minimal Size** *behave as if the ports were visible.*

To show/hide ports:

The representation of a component can be modified by means of the context menu **Ports**. Ports of a component can be added or removed by the method this way.

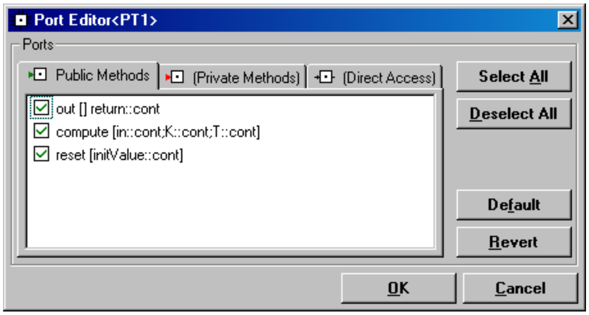
Note

The ports of a method can only be added or removed together.

1. Public Methods

   - Select **Ports** → **Methods** from the context menu of the occurrence.

     The port editor opens; the "Public Methods" tab is displayed. It contains a list of all public methods of the added component.
     The selected methods are shown with a tick beside them.



Port Editor<PT1>

Ports

Public Methods | (Private Methods) | (Direct Access)

- ✓ out [] return::cont
- ✓ compute [in::cont;K::cont;T::cont]
- ✓ reset [initValue::cont]

2. Private Methods

   - Select **Ports** → **Methods** from the context menu of the occurrence.

- Select the "Private Methods" tab.

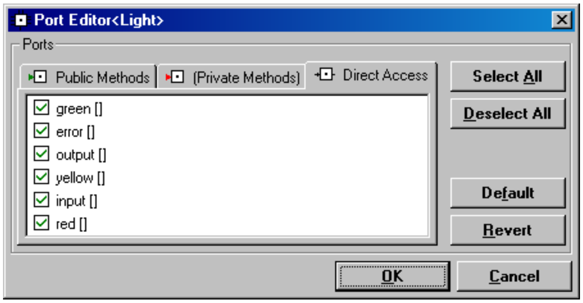
3. Direct Access

- Select **Ports** → **Direct Access** from the context menu of the occurrence.

  The port editor opens; the "Direct Access" tab is displayed. It contains a list of all direct access methods of the added component (e.g. the inputs/outputs of a state machine)

or

- select **Ports** → **Methods** from the context menu of the occurrence.
- Select the "Direct Access" tab.


Port Editor<Light>

Ports

Public Methods | (Private Methods) | Direct Access

- [x] green []
- [x] error []
- [x] output []
- [x] yellow []
- [x] input []
- [x] red []

4. Editing

- Double-click on an entry not marked to select it.
- Click on **Select All** to select all entries.
- Double-click on a marked entry to deselect it.
- Click on **Deselect All** to deselect all entries.
- Click on **Revert** to undo all changes made so far.
- Click on **Default** to restore the setting specified in the layout editor.
- Activate one of the tabs to make further settings

or

- click on **Cancel** to cancel the action

or

- confirm your selection with **OK**.

The ports of the marked methods/processes are displayed in the occurrence; those of the methods/processes not marked are removed.

If you remove a port which is connected to another element, the connecting line is removed with it.

The positions for newly added ports are determined automatically. Inputs are created on the left-hand side; outputs on the right. If you want to add ports for which the current size of the occurrence has no space, the layout is enlarged automatically.

To use changes as a new default layout:

If you want to use the modified layout of an occurrence as the new default layout, proceed as follows.

- Select **Use as Default Layout** from the context menu of the occurrence.

  The following query is displayed:

  ```
  Do you want to use the current
  layout as the default layout for
  all future occurrences of this
  class?
  ```

- Cancel with **Cancel**

or

- confirm with **OK**.

  The modified layout is now the new default layout. It is available in the layout editor of the component.

  If you create new occurrences of the component, the new default layout is used. Existing occurrences in diagrams are *not*, however, updated.

To restore the default layout:

As long as you have not used the command **Use as Default Layout**, you can restore the default layout defined in the layout editor of the component.

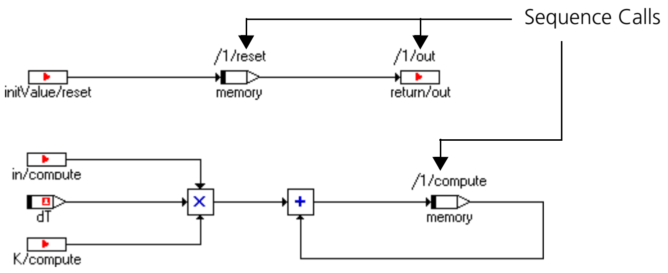
- Select **Default Layout** from the context menu of the occurrence.

  The default layout defined in the layout editor of the component is restored.

  If connected ports are removed in this process, the connecting lines are removed with them. If connected ports are moved, the connecting lines are moved with them. They are retained.

4.1.6 Sequence Calls

A sequence call is linked to every assignment operation or every method call of an included component. Every sequence call represents an instruction in an ASCET diagram. Sequence calls determine the control flow in diagrams by assigning every instruction to a method and determining the order of the instructions within a method.

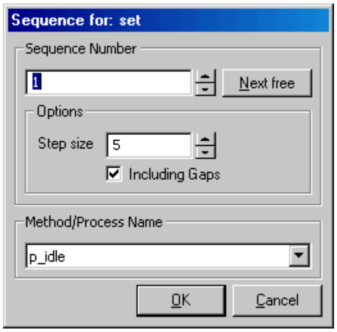


In the Block Diagram Editor, connecting lines between elements and/or operators are shown as colored lines as long as the sequencing for the instruction or operation linked to the element has not been resolved. The color indicates that the sequencing still has to be resolved. All lines to which a sequence call has been assigned are shown in black.

You can edit the sequence calls individually or in groups, manually or automatically.

Editing Individual Sequence Calls

The Sequence Editor is used to edit and configure individual sequence calls.



This contains the following elements:

- The "Sequence Number" field – this is where you can enter a number for the sequence call.

  By default, the number 1 is shown for new sequence calls and the current number for sequence calls which have already been assigned.

- The **Next free** button – this is used to set the next free number in accordance with specific rules (see page 236).

- The "Step size" box – this is where the step size for the search for the next free number is set.

  The value of the "Sequence Step Size" option from the ASCET option window (see ""Sequencing" node" on page 56) is already set by default. If you change the value in the Sequence Editor, the value in the option window is adjusted accordingly.

- The **Including Gaps** option – this is where it is determined whether gaps should be included between existing sequence numbers or not when using **Next free**.

  The value of the **Use Gaps** option is preset from the ASCET option window (see ""Sequencing" node" on page 56). If you change the setting of the **Including Gaps** option in the Sequence Editor, the setting is adjusted in the option window.

- The "Method/Process Name" combo box – this is where the sequence call is assigned to a process or method.

  The process or method selected in the "Diagrams" area of the Block Diagram Editor is preset for new sequence calls. The current process or method is displayed for assigned sequence calls.
  If you change the selection in the Sequence Editor, the selection in the "Diagrams" area is also changed.

- **OK** accepts the changed settings and closes the Sequence Editor; **Cancel** closes the Sequence Editor without the changes being accepted.

To edit the sequence call in the Sequence Editor:

- Right-click a sequence call in the drawing area to open the context menu.
- Select **Edit** from the context menu to open the Sequence Editor.
- From the "Method/Process Name" combo box, select the process/method for the sequence call.
- In the "Step size" field, enter the step size for automatic determination of the sequence number.
- Activate the **Including Gaps** option if gaps between existing numbers are to be taken into consideration in the automatic determination of sequence numbers.
- In the "Sequence Number" field, set the number of the sequence call

or

- click **Next free** to set the next free number for the selected process/method.
- Click **OK**.

  A check is carried out to see whether the set combination of number and process/method has already been assigned. If not, the combination is assigned to the sequence call and displayed in the block diagram.
  "Step size" and **Including Gaps** are accepted in the ASCET option window.

The following rules apply for determining the number using **Next free**:

1. Only whole multiples of the value in the "Step size" field (corresponds to "Sequence Step Size" in the "Sequencing" node of the ASCET option window, see page 56) are taken into consideration. If, for example, the value 5 is set, only the numbers 5, 10, 15, 20, ... are checked.
2. If the **Including Gaps** option is activated, any gaps between existing sequence numbers are filled. The first condition still applies; gaps which are not whole multiples of the value in the "Sequence Step Size" box are not filled.

   Example:

   If, e.g., the numbers 1–3, 5–9 and 11 have already been assigned and 5 has been specified in the "Sequence Step Size" box, 10 is assigned, not 4.

Note

*The system saves the sequence number last assigned. Gaps below this number are **not** filled!*
When you leave the editor or reset one (page 239) or several sequence calls (page 243), the number saved is reset; automatic numbering starts again at 1.

3. If the **Including Gaps** option is not activated, a search is carried out for the next whole multiple of the value in the "Sequence Step Size" field after the highest available number.

   In the above example (1–3, 5–9 and 11 assigned), 15 is assigned.

To assign individual sequence calls automatically:

You can edit individual sequence calls simply and quickly as follows:

- In the "Diagrams" area, select the process/ method to which the sequence call is to be assigned.
- In the drawing area, double-click the sequence call you want to edit,

or

- right-click the sequence call.

- Select **Next Number** from the context menu.

  In both cases, you can select either a sequence call which has not yet been assigned or one which has already been edited.

Note

If no method/process has been selected, the Sequence Editor is opened with a double-click and the **Next Number** *context menu.*

The selected process/method as well as the next free number are assigned to the sequence call.

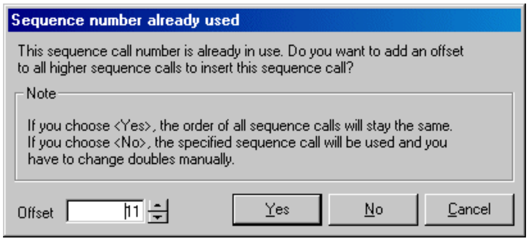
The rules described on page 236 apply, using the values for "Sequence Step Size" and **Use gaps** set in the ASCET option window.

To increment/decrement individual sequence calls:

- Right-click the sequence call whose number you want to edit.
- Select **Change** → **Increment** from the context menu to increase the sequence number of the call by 1.
- Select **Change** → **Decrement** to decrease the sequence number of the call by 1.

To use numbers already assigned:

If you want to assign an existing combination of process/method and number in the Sequence Editor or when incrementing/decrementing, the following warning is displayed.


Sequence number already used

This sequence call number is already in use. Do you want to add an offset to all higher sequence calls to insert this sequence call?

Note

If you choose <Yes>, the order of all sequence calls will stay the same.
If you choose <No>, the specified sequence call will be used and you have to change doubles manually.

Offset 1

It is possible to shift the existing number as well as all higher numbers by an offset which can be defined. The order of the shifted sequence calls is retained.

- In the "Offset" field, set the value for the offset of the sequence calls.

  The value of the "Sequence Shift Offset" option from the ASCET option window (see ""Sequencing" node" on page 56) is preset.

- Click **Yes**.

  The edited sequence call is assigned the values set in the Sequence Editor. The other sequence call as well as the sequence calls with higher numbers which belong to the same process/method are shifted by the value specified under "Offset".

Or

- Click **No** to assign the existing combination anyway.

  This can be useful if the first occurrence of the combination was a mistake, e.g. if the wrong number was entered.

Note

You then have to edit the double sequence calls manually to resolve the conflict.
Otherwise corresponding error messages are created during code generation.

Or

- Click **Cancel** to return to the Sequence Editor.
- Set a combination which has not yet been assigned.

To reset an individual sequence call:

Proceed as follows to reset an individual sequence call:

- Right-click a sequence call.

- Select **Change** → **Reset** from the context menu.

  The current values of the sequence call are reset; the connecting line is again shown colored.

  At the same time, the number last assigned saved internally is deleted so that automatic assignments start again at the lowest possible value.

To move between sequence calls:

- Select a sequence call in the drawing area.
- Select **Sequence Call** → **Next Seq. Call** to select the next call in the sequence.
- Select **Sequence Call** → **Last Seq. Call** to select the previous call in the sequence.

To change the visibility of individual sequence calls:

- Right-click the input port of the element to which the sequence call is linked.
- Deactivate the **Sequence Call** command in the context menu.

Or

- Right-click the sequence call.
- Select **Hide** from the shortcut menu.

  The sequence call is hidden.
- Right-click the input port of the element to which the sequence call is linked.
- Activate the **Sequence Call** command in the context menu.

  The sequence call is displayed.
- Select **Select Complete Port** to mark the port to which the sequence call is connected.

  The input port of the relevant diagram element is shown in blue. This feature is useful to be able to follow sequence calls in complex diagrams.

To create a sequence of protected sequence calls:

- Select **Atomic** → **Start** to start a sequence of protected sequence calls.

  A sequence of protected sequence calls cannot be interrupted in a real-time environment.
- Select **Atomic** → **Stop** to end a sequence of protected sequence calls.

Editing Several Sequence Calls

A large number of sequence calls may be necessary in complex diagrams. In this case it is useful to be able to edit the sequence calls in groups. You can change all sequence calls of selected blocks, individual processes or methods or a complete diagram.

Note

Make sure you ***only*** *select* ***sequence calls****. If connectors (see "Connectors" on page 244) are selected too, the automatic assignments do not work. If only connectors are selected, only* **Sequence Calls** → **Sequencing - Ignore Info** *works.*

To assign sequence calls automatically:

- In the "Diagrams" area, select the process/method to which the sequence calls are to be linked.
- Select all elements to which you want to assign sequence calls.

  Sequence calls can only be assigned automatically to one process/method at a time. Therefore, you should only select elements which are to be linked to the same process or method.

  Note

  If you do not select any elements, there may well be incomplete and incorrect assignments if there are several processes/methods.
- Select **Sequence Calls** → **Sequencing - Ignore Info**.

- This command analyzes the diagram and assigns the sequence calls in accordance with the integrated sequencing algorithm.

The automatic sequencing algorithm requires a procedure based on the data flow, i.e. the elements are sequenced in the order defined by the data flow. If the diagram is to be sequenced in another order, the sequence calls have to be set manually.

To automatically assign sequence calls from a specific number:

- In the "Diagrams" area, select the process/ method with which the elements are to be linked.
- Select all elements to which you want to assign sequence calls.
- Select **Sequence Calls** → **Sequencing Starting With**.
- Enter a number in the input box.

  The selected elements are sequenced with the selected process/method, starting with the specified sequence number.

To add sequence calls to an existing sequence:

It is possible to add sequence calls to a sequence which was defined earlier, i.e. to a number of sequence calls which have already been assigned to a process/ method. In this case, the first of the newly assigned sequence calls receives a number which is one higher than the last one in the sequence defined earlier.

- Select all elements which you want to add to a sequence which has already been defined.
- In the "Diagrams" area, select the method to which the elements are to be linked.
- Select **Sequence Calls** → **Sequencing Appending**.

  The selected sequence calls are appended to the defined sequence for the selected process/ method.

To shift several sequence calls:

Proceed as follows if you want to shift the sequence numbers of a group of sequence calls:

- Right-click the sequence call with the lowest number.
- Select **Change** → **Shift by offset** from the context menu.

  The sequence number of the call and all calls with a higher number which are linked to the same method or the same process are offset upwards with the value set in the "Sequence Shift Offset" option in the ASCET option window (see ""Sequencing" node" on page 56).

To scale sequence calls:

It is possible to scale all sequence calls of a process/method or the entire diagram.

- From the "Diagrams" area, select the process/ method whose sequence calls you want to scale.
- Select **Sequence Calls** → **Scale To Step Size** → **For Method/Process**.

  The sequence calls of the method/process are scaled in accordance with the value entered under "Sequence Step Size" in the ASCET option window (see page 56).

Or

- Select **Sequence Calls** → **Scale To Step Size** → **For Diagram** to scale all sequence calls of the diagram.

To reset several sequence calls:

- Select **Sequence Calls** → **Reset** → **For Diagram** to reset all sequence calls of the current diagram.
- Select **Sequence Calls** → **Reset** → **For Method/Process** to reset all sequence calls assigned to the method/process which is currently selected in the "Diagrams" areas.

- Select **Sequence Calls** → **Reset** → **For Selected Blocks** to reset all sequence calls which are assigned to the elements currently selected in the drawing area.

  All commands require a confirmation and then the relevant sequence numbers are set to 0 and the sequence names are deleted. The connecting lines are again shown colored.

To change the visibility of several sequence calls:

- Select **Sequence Calls** → **Hide** → **For Diagram** to hide all sequence calls in the current diagram.
- Select **Sequence Calls** → **Hide** → **For Method/Process** to hide all sequence calls which are assigned to the method/process currently selected.
- Select **Sequence Calls** → **Hide** → **For Selected Blocks** to hide all sequence calls which are assigned to the elements currently selected in the drawing area.
- Select **Sequence Calls** → **Hide** → **Unused** to hide all sequence calls which are not currently being used.

  The **Show** command reverses the effect of the **Hide** command with the same four options being available.

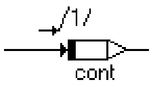
Connectors

Connectors are used to connect an assignment with a control flow instruction such as `If...Then` or `If...Then...Else`.

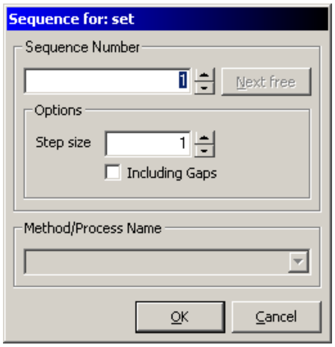
To create connectors:

- Right-click the sequence call you want to make into a connector.
- Select **Connector** from the context menu.

  The sequence call becomes a connector. This can be connected with a control flow instruction.



Connectors can be edited like sequence calls in the Sequence Editor (see "To edit the sequence call in the Sequence Editor:" on page 236). The **Next free** button and the "Method/Process Name" field are, however, deactivated. Double-clicking the connector and the shortcut menu **Next Number** also open the Sequence Editor.



Most possibilities for automatic editing are *not* available for connectors.

To assign connectors automatically:

Note

This is the only automatic assignment which works for connectors.

- In the "Diagrams" area, select the process/method with which the sequence calls are to be linked.
- Select all connectors which are to be used in the selected process/method.

Note

*Make sure you **only** select **connectors** as automatic assignment does not work otherwise.*

- Select **Sequence Calls** → **Sequencing - Ignore Info**.

  This command analyzes the diagram and assigns the numbers of the connectors in accordance with the integrated algorithm.

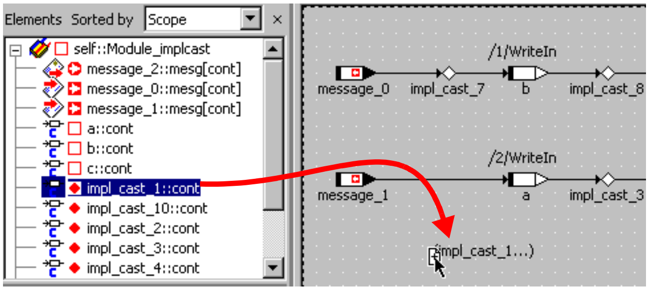
4.1.7 Implementation Casts in Block Diagrams

In the block diagram editor, implementation casts (see "Implementation Casts" in the ASCET Reference Guide) can be inserted in the same way as all other elements using the relevant button in the button bar (here: **Implementation cast**,). Once generated, they can be added to the drawing area from the element list by Drag&Drop and can be connected there in the same way as all other elements.

Note

Implementation casts ***cannot*** *be applied to logical elements. If you connect an implementation cast to a logical element, the connecting line is shown in red to indicate the error.*

There are no sequence calls for implementation casts; the correct order is determined from the context by code generation.



In the block diagram editor, there is another very convenient way of adding implementation casts. This is particularly useful for existing arithmetical calculation chains. Using the context menu of the arithmetic operators `+`, `-`, `*`, `/`, `abs` and `neg` you can add implementation casts automatically for all inputs and outputs of the operation by selecting **Add Implementation Casts**.

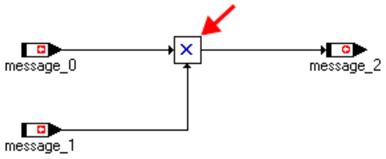
To add implementation casts to operators automatically:

Note

The procedure only works if ***none*** *of the inputs and outputs of the operator is directly connected to an implementation cast. Otherwise the following error message appears:*

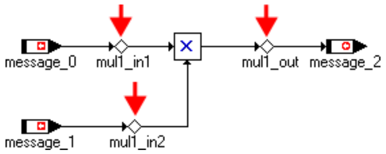
```
This operator is already connected to at least one
implementation cast. Please specify further implemen-
tation casts individually.
```

- Activate the operator which is to have implementation casts added to it.



- Click the operator with the right-hand mouse button and select **Add Implementation Cast** from the context menu.

  If there is sufficient space in the drawing area, implementation casts are added to all connections with inputs and outputs of the operator.



  If there is not sufficient space, the following warning appears:

```
The layout is too tight to place an implementation
cast automatically. Please rearrange the diagram.
```

- Lengthen the connections to the operator and try again.

The implementation casts are named automatically in accordance with the following scheme:

```
<operator><m>_<pin type><n>
```

- `<operator>` can, depending on the selected operator, have the values `add`, `sub`, `mul`, `div`, `abs` or `neg`.
- `<m>` is the number of the operator. The first operator of a type for which implementation casts are generated in this way is assigned the number `1`; further operators of the same type are then numbered consecutively (2,3,....).
- `<pin type>` is the description of the operator pin connected to the implementation cast, i.e. `in` for inputs and `out` for outputs.
- `<n>` is the number of the implementation cast. Implementation casts connected tio the inputs and output of the operaor are nnumbered separately.
  - The implementation cast connected to the first operator input is assigned the number `1`; further inputs are numbered consecutively. The number is omitted if the operator has only one input.
  - If the operator output is connected to more than one element, the implementation casts are numbered, beginning with `1`.
    If the operator input is connected to one element, the number is omitted.

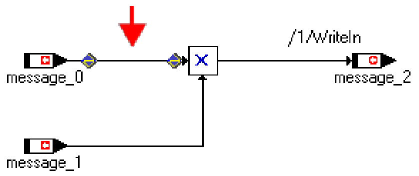
To add implementation casts to a connection automatically:

You can also add an implementation cast to all connecting lines (data paths) of arithmetical values via the context menu.

Note

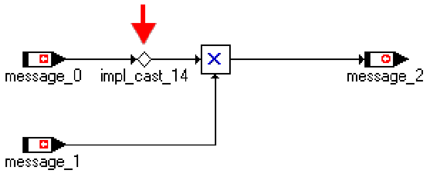
*This is **not** the case for connections with other operators than* `+`, `-`, `*`, `/`, `abs`, `neg`, `max`, `min`, *and* `mux`, *connections to to logical elements or control flow connecting lines.*

- Select the connection you want to add an implementation cast to.



- Right-click on the connection and select **Add Implementation Cast** from the context menu.

  An implementation cast is automatically added to the connection.



The implementation casts are named automatically in accordance with the following scheme:

```
impl_cast_<n>
```

`<n>` is the number of the implementation cast. The first implementation cast created automatically on a connection is not assigned a number; the second one is assigned the number `1`, further implementation casts created automatically on any connections are then numbered accordingly.

This scheme applies to all implementation casts which were *not* created automatically for an operator (see page 247).

To show/hide the display of implementation casts:

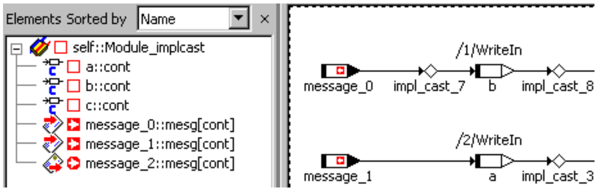
By default, the implementation casts are displayed like all other elements in the "Elements" list. You can, however, hide the display.

- Open the submenu **View** → **Show/Hide**.

  If the display is activated, **Impl. Casts in Elements List** is marked with a tick.

- Select **View** → **Show/Hide** → **Impl. Casts in Elements List**.

  The implementation casts are no longer displayed in the "Elements" list; the tick beside the menu function is removed.

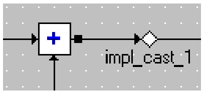


  In the drawing area, the implementation casts continue to be displayed in accordance with the settings for the current view. The implementation casts context menu, however, only contains a few functions.

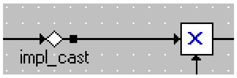
- Select **View** → **Show/Hide** → **Impl. Casts in Elements List** again to reactivate the display.

Implementation casts can be used together with temporary variables. Please note the following:

- If an implementation cast follows a temporary variable, the temporary variable is not influenced by the implementation cast.



- Temporary variables immediately after an implementation cast are supported.



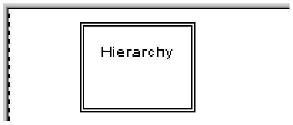
4.1.8 Graphical Hierarchies

To organize complex diagrams more clearly, you can group parts of a diagram into a hierarchy block, which is then only visible as a symbol on the top level of the diagram. Whether any diagram item is inside a hierarchy block or not, does not in any way affect its functionality, hierarchies serve only for graphical structuring. Hierarchies can be nested so that hierarchy blocks can contain other hierarchy blocks.

To add a new hierarchy:

- Click on the **Hierarchy** button on the toolbar of the block diagram editor to load the mouse cursor with a hierarchy.
- Click inside the drawing area where you want to position the hierarchy.

  The hierarchy block is added to the diagram.



To convert diagram elements into a hierarchy block:

- Select the diagram items you want to put in a hierarchy block.
- Click on the **Hierarchy** button.

  A hierarchy block that contains the selected elements is added to the diagram.

The selected diagram items are placed inside a newly created hierarchy block. A pin with a default name is created for each line that connects a diagram item outside the hierarchy frame with a diagram item inside. It is not possible to move diagram items into a hierarchy block directly, e.g. by drag-and-drop. They always have to be copied via the clipboard or moved as described here.

To move elements into or out of a hierarchy frame:

- Use **Cut** or **Copy** to copy/move diagram elements from the higher level to the clipboard.
- couble-click on the hierarchy that is to contain the elements.

  The hierarchy content is displayed.
- Use **Paste** to insert the diagram elements.
- Double-click in the drawing area to return to the higher level.

Resolving a hierarchy block:

- right-click on the hierarchy block and select **Resolve** from the context menu.

  The hierarchy block is removed and all diagram items it contains are brought into the diagram one level up.

Or

- Select a hierarchy block.
- Select **Edit** → **Delete**.

  The hierarchy block, and all diagram elements it contains, are deleted.

To change the appearance of a hierarchy block:

- Right-click inside the block and select **Rename Hierarchy,** from the context menu to rename the hierarchy block.

  You are prompted for a name for the hierarchy.
- Type the name into the prompt box and click **OK**.
- Right-click inside the block and select **Show/ Hide Name** from the context menu.

  This hides the name. Select the command again to bring it back.
- Right-click inside the block and choose **Change Icon** to assign an icon file to the hierarchy.

  A file prompter is displayed from which you can select a bitmap image file in either TIFF, PCX or BMP format.
- Select the image file you want and click **OK**.

  The image file is stored with the diagram in the ASCET database.
- Select the hierarchy block and resize the frame by dragging the sizing handles.

- Select **Set To Default Size** from the context menu to revert to the default size of the hierarchy block.

The technique for navigating between hierarchy levels is the same as that for navigating between included components of a diagram.

To navigate between hierarchy levels:

- Double-click on the hierarchy block

or

- right-click inside the block and select **Next Level** from the context menu.

  You are inside the hierarchy block, the content of the current block is shown in the drawing area.

- Inside the hierarchy, double-click on the drawing area (not on a diagram item).

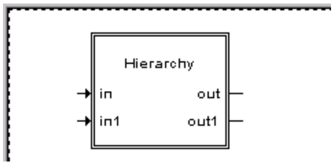
  You leave the hierarchy and the block you moved from is selected.

To add input and output pins to the hierarchy:

Data flow between the various hierarchy levels works via input and output ports. These are simply connection lines that extend across the levels.

- Right-click on the hierarchy block.
- Select **Add Inpin** or **Add Outpin** from the context menu.

  You can add any number of input and output pins to the hierarchy frame. The input and output pins are represented by arrow symbols containing the pin name inside the hierarchy block.



To change the appearance of input and output pins:

- Right-click on an input or output pin.

- Select **Rename Pin** from the context menu to change the name of the pin.
- Type the name into the prompt box and click **OK**.
- Select **Pin Name** to hide the pin name.
- Select the command again to reveal it.
- Select **Remove Pin** to remove the input or output pin.
- Right-click on the hierarchy block and select **Show Pin Names** or **Hide Pin Names** to show or hide all the pin names for the hierarchy block.

When you connect a diagram item to a pin, and then connect the pin to another item inside the hierarchy block, it is equivalent to connecting the two elements directly.

4.1.9 Navigating Between Diagrams

Components with Multiple Diagrams

A component specification for a class or module can consist of more than one diagram. This feature is useful for structuring complex specifications. A diagram can either be public, i.e. contain only public methods, or private, i.e. contain only private methods.

Public methods can be accessed from other components, private methods cannot. Private methods can only be accessed from inside the component. Private diagrams containing actions and conditions are a special case. These are used in state machines only.

Each component specified as block diagram contains at least one public diagram named `Main`. Classes can have any number of public or private diagrams, whereas modules only have public diagrams.

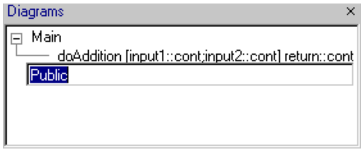
To create a new diagram:

- Select **Diagram** → **Add Diagram** → **Public** or **Private**

or

- in the "Diagrams" pane, select **Add Diagram** → **Public** or **Private** from the context menu.

  A new public or private diagram is added in the "Diagrams" pane. The name of the new diagram is selected automatically.

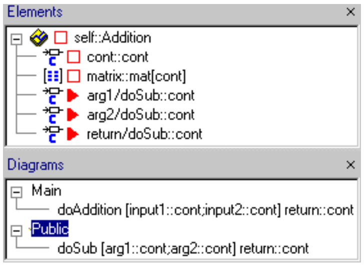


Diagrams
Main
doAddition [input1::cont;input2::cont] return::cont
Public

- Type in a name and press <ENTER>.

  You can also rename the diagram later. To do so, select **Diagram** → **Rename Diagram**
- Add the methods or processes necessary to specify the functionality of the diagram.

In the "Elements" list, the interface items, i.e. arguments, local variables and return value, are displayed only for the methods in the current diagram. Other elements, such as variables or parameters can be used across diagrams and are always visible in the "Elements" list.



Elements
self::Addition
cont::cont
matrix::mat[cont]
arg1/doSub::cont
arg2/doSub::cont
return/doSub::cont

Diagrams
Main
doAddition [input1::cont;input2::cont] return::cont
Public
doSub [arg1::cont;arg2::cont] return::cont

In the block diagram editor, you work on one diagram at a time. If a component contains more than one diagram, you can switch between diagrams by loading a new diagram into the drawing area.

To load a diagram:

- Select a diagram.
- Select **Diagram** → **Load Diagram**

or

- select **Load Diagram** from the context menu.

  The diagram is loaded in the drawing area.

  If there are unsaved changes in the current diagram, you are prompted to save the current diagram before the new diagram is loaded.

To delete a diagram from a component:

- In the "Diagrams" list, select the diagram you want to delete.
- Select **Diagram** → **Delete Diagram**

or

- select **Delete Diagram** from the context menu.

  The diagram is deleted from the component, together with all its methods/processes.

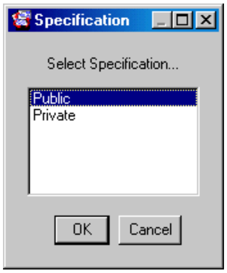
Note

The first diagram in the "Diagrams"pane cannot be deleted.

To move methods between diagrams:

- In the "Diagrams" pane, select the method you want to move.
- Select **Diagram** → **Move** to move the method.

  The "Specification" window displays a list of all the diagrams for the current component.



- Select the diagram to which you want to move the method and click **OK**.

  The selected method is moved to the target diagram. Any sequence calls present in the original diagram for the method are reset.

You can move methods only if their interface elements have no occurrences in the diagram. Otherwise, remove the occurrences of interface elements first and then move the method.

Navigating Between Components

When you are editing a component that includes other components in the block diagram editor, you can edit those sub-components without having to go through the Component Manager to open the editor on an included component.

To navigate between different levels of a block diagram:

- In the drawing area or "Elements" pane, select the component you want to edit.
- Select **Element** → **Edit Component**

or

- select **Edit Component** from the context menu of the component

or

- double-click on the included component.

  An editor opens for the selected component. If the including component has unsaved changes, a new editor window opens. Otherwise, the sub-component is loaded into the original editor window.
- To navigate upwards in a containment hierarchy, double click in the drawing area, not on a diagram item.

  You are moved back up one level. If you have modified the sub-component, a new editor window opens for the previous component; the window for the sub-component remains open.

The mechanism described here is also used for navigating between different levels of graphical hierarchies in the same component.

4.1.10 Analyzing Components

After you have created a block diagram, you will usually want to experiment with it to see whether it works as intended. This section describes how to prepare for experimenting with a component, and how to start the experimentation environment.

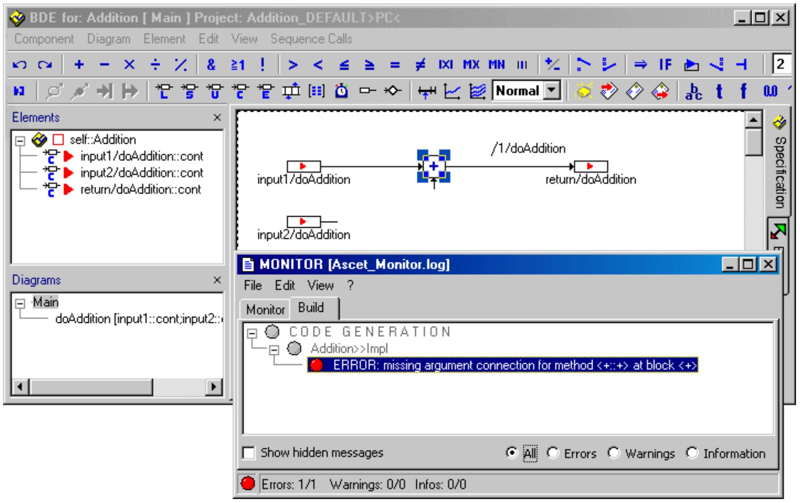
Note

For a single ASCET module, code can be generated and simulated without project context only with the `Physical Experiment` *code generator (see page 371). For the other code generators the module must be integrated into a project. A so-called default project can be defined for each class or module for that purpose. This is the only way to access the implementation information. Without project context, the conversion formulas as well as all implementations of imported entities are missing.*

To analyze a diagram:

- Select **Diagram** → **Analyze Diagram** to analyze the current diagram.

  The diagram is checked for syntactic errors. Unassigned sequence calls, missing connections etc. are reported in the monitor window (see chapter 2.4 "The ASCET Monitor Window").



CODE GENERATION
Addition>>Impl
ERROR: missing argument connection for method <+::+> at block <+>

Errors: 1/1 Warnings: 0/0 Infos: 0/0

- Click on an error message in the monitor window, "Build" tab, to have the error highlighted automatically in the block diagram editor.

To generate code for a component:

- Select **Component** → **Generate Code**

or

- click on the **Generate Code** button.

  The C code for the current component is generated.
  The system will display any error messages returned by the Code Generator. Again, you can find the relevant diagram item by clicking on an error message.

When the code for a component has been generated successfully, you can open the experimentation environment for the component.

The code generated by ASCET is stored in the database, where it cannot be viewed by the user. It is, however, possible to write the code to a file where it can be viewed with any text editor.

To view the generated code:

- Generate the code for the current component as described above.
- Select **Component** → **File Out Generated Code** → **Flat** or **Recursive** to store the code generated on the file system.

  The files generated are written to the directory selected. If code has been stored recursively, a separate file is created for each component. The names of the files generated are listed in the ASCET monitor window.
- Open the files with a text editor to view them.

If you select **Flat**, only the code for the currently selected component will be filed out; if you select **recursive**, the code for the components referenced by the current component will also be written to a file.

Another way of viewing code is to select a user defined external editor from ASCET (e.g. Notepad, Codewright, etc.), or an internet browser, respectively. As the external editor is connected via a file system, files with the suffixes "`*.c`" and "`*.h`" must be associated with the editor in the Windows configuration.

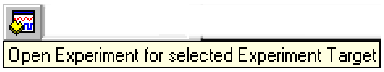
- Select **Component** → **View Generated Code**.

  C code is generated and displayed in an external editor.
  The text editor can be selected in the ASCET options window, "ASCII Editor" node (cf. page 60).

To start an offline experiment:

- Select **Component** → **Open Experiment**

or



Open Experiment for selected Experiment Target

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

  Code is generated and compiled with the compiler specific to the current target, and the experimentation environment for the component is opened. The generated files are stored in the `cgen` directory.

The experimentation environment is described in "The Experimentation Environment" on page 561.

Under some circumstances, the global elements (see "Defining Global Communication" on page 388) are not updated properly in the default project. The following error message is displayed in the ASCET monitor window:

```
Error: need export for imported element <name> with
type <type>
```

To correct this error, proceed as follows:

To define global elements in the default project:

- In the block diagram editor, select **Component** → **Default Project** → **Resolve Globals** to resolve the global elements.

- Select **Component** → **Default Project** → **Delete Unused Globals** to delete unused global elements.

  You can now restart the experiment.

4.1.11 Data Exchange

A data set is always associated with the component it came from. To import the data set, use the command **File** → **Import** in the Component Manager. This command writes the imported data set back to the component it belongs to. The same rules apply to exporting data sets as to exporting database items. See chapter 2.3.3 "Exporting Folders and Database Items" on page 89 for details.

To export the data set of a component:

- Select **Component** → **Export Data**.

  The file selection dialog box opens.
- Select a file name and a path name.
- Click on **OK**.

  The data set is written to the file selected.

It is possible to store parts of a diagram in a file and import them into another component. All diagram items except interface elements can be stored in a file.

To store parts of a diagram in a file:

- Select the part of the diagram you want to write to a file.
- Select **Edit** → **Copy**.
- Select **Edit** → **File Out Buffer**.

  The file selection dialog box opens.
- Select a path and type in a file name with the extension `.asc`.
- Click on **OK**.

  The diagram items on the clipboard are stored in the file specified.

To read a diagram part from a file:

- Select **Edit** → **File In Buffer** to load the diagram into another component.

  The file selection dialog box opens.

- Select the file that contains the diagram elements you want to import.
- Click on **OK**.

  The imported diagram parts are copied to the database clipboard.
- Select **Edit** → **Paste**.

  The diagram items from the file are written into the current diagram of the current component.

It is possible to write the data from a table or an array to a file and to read it in again. The data is written in tab-delimited ASCII format, so it can be read and edited with any spreadsheet or word processor.

To write the data from an array or a table to a file:

- Select the table or array in the "Elements" pane.
- Select **Element** → **File Out Data**.

  The file selection dialog opens.
- Select a path and a filename with the extension `.dat`.
- Click on **OK**.

  The data from the table or array selected is written to the file.

To read the data for an array or a table from a file:

- Select the table or array in the "Elements" pane.
- Select **Element** → **File In Data**.

  The Windows file selection dialog box opens.
- Select the file that contains the data you want to read in.
- Click on **Open**.

  The data in the file is written to the selected table or array.

Saving Diagrams

Saving a component specification:

To save the specified component, proceed as follows:

- Select **Diagram** → **Store to Cache**

or

- in the "Diagrams" pane, select **Store to Cache** from the context menu.

  The component is saved in the cache. This command does not save the component permanently in the database.

- To make the changes permanent, select **File** → **Save Database** in the Component Manager.

or

- click on the **Save** button.

  The **Save** command stores all the changes you have made in the database.

Viewing and Printing Diagrams

To change the way a block diagram is displayed:

- Select **View** → **Redraw** to redraw the diagram.

  Sometimes a diagram may become corrupted when you move elements around. Choosing this command will clean up the diagram.

- In the "Zoom" combo box, select a value to scale the diagram.

  You can also enter any value directly.

- In the "Zoom" combo box, select `Page Layout` to view one print page.

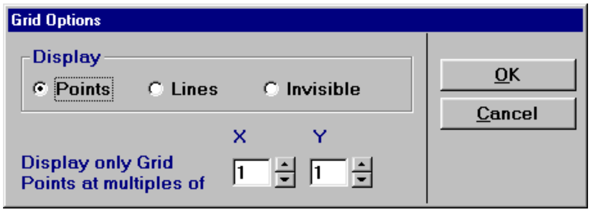
  The diagram is zoomed out so that the first print page is displayed in its entirety. Proportions are kept.

- In the "Zoom" combo box, select `100%` to return to the default size.

To change the grid in the drawing area:

- Select **View** → **Grid**.

  The "Grid Options" dialog box opens.



Display only Grid Points at multiples of: X 1, Y 1

Display: [✓ Points] Lines Invisible

- Click on the **Points** radio button to display grid points in the drawing area.
- Click the **Lines** radio button to display grid lines.
- Click the **Invisible** radio button to turn off the grid display.
- Adjust the grid multiples at which the points or lines are displayed.

  If, for instance, the **Points** option is selected and the display is set to two, the distance between the grid points will be twice the default.
- Click **OK** to set the grid options.

To set up the printing area:

- In the ASCET options window, "Paper Size" node (cf. page 57), select the paper size.

  The settings becomes effective the next time you open the block diagram editor.
- Select **View** → **Page frame Portrait** to print the diagram in portrait format.

- Select **View** → **Page frame Landscape** to print the diagram in landscape format.

  The default for the orientation is set in the ASCET options window, "Paper Size" node (cf. page 57), "Paper Orientation" option.

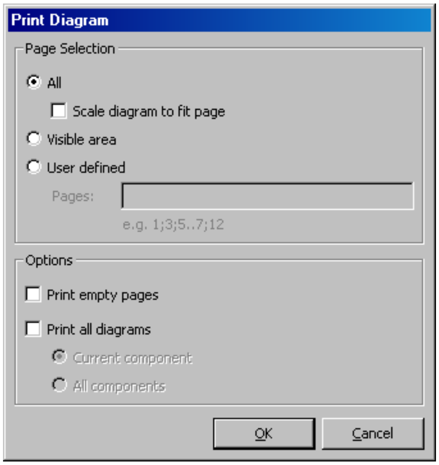
When the drawing area exceeds the selected page format, print pages are marked by dashed lines in the drawing area. In adition, you can use the ASCET options dialog ("Block Diagram" node (cf. page 54), **Show Page Number**) to switch on the display of page numbers in the block diagram editor. Dashed lines and page numbers can be covered by diagram elements.

The color of lines and page numbers is set in the ASCET options window, "Colors" node (cf. page 55), "Watermark Color" option.

To print a block diagram:

- Select **View** → **Print Diagram** to print the drawing area.

  The "Print Diagram" window opens.



Print Diagram

Page Selection

- [✓ All]
  - ☐ Scale diagram to fit page
- Visible area
- User defined
  - Pages:
  - e.g. 1;3;5..7;12

Options

- ☐ Print empty pages
- ☐ Print all diagrams
  - [✓ Current component]
  - All components

OK Cancel

- In the "Print Diagram" window, set the print options.
- Click **OK**.

  The "Printer Selection" window opens.
- In the "Printers" field, select a printer.

  Use **Setup** to change the printer settings.

- Click **OK** to accept the selection.

  The component is printed according to your settings.

The "Print Diagrams" window contains the following options:

- **All**

  Prints the entire drawing area.

- **Scale diagram to fit page**

  Scales the diagram so that it fits on one print page. Only available if **All** is activated.

- **Visible area**

  Prints the visible part of the drawing area.

- **User defined**

  Prints a user-defined part of the drawing area.

  Note

  If **User defined** *is activated, the* **Print empty pages** *and* **Print all diagrams** *options are without effect.*

- "Pages:"

  Input field for the pages to be printed. A ; separates individual pages, start and end of a page range are separated by two dots (..). Only available if **User defined** is activated.

- **Print empty pages**

  Empty pages are printed.

- **Print all diagrams**

  All diagrams of the component are printed.

- **Current component**

  Only the diagrams of the current component are printed. Only available if **Print all diagrams** is activated.

- **All components**

  The diagrams of included components are printed, too. Only available if **Print all diagrams** is activated.

Using Referenced Components

There are several commands that apply to referenced components, i.e. those included within the current component. These have the same function as the commands in the **Component** menu, but they affect only the included component currently selected.

To edit an included component:

- In the "Elements" pane, select an included component.
- Select **Element** → **Edit Component** to open the block diagram for the component selected.

  This command has the same effect as double-clicking on the referenced component.
- Select **Element** → **Notes** to view the notes for the imported component.

  When you select this command, you get a read-only view of the notes associated with the component, i.e. you cannot edit them. Views are explained in section "Notes" on page 693.
- Select **Element** → **Export Data** to export the data set of the referenced component.

  This command has the same effect as **Component** → **Export Data** but relates only to the referenced component selected in the "Elements" list.

4.2 The State Machine Editor

A *state machine* consists of a state diagram and additional normal block diagrams. The state diagram is a special block diagram, where the states are represented as rectangles with rounded corners, and the transitions by directed arcs. Every state can be a *hierarchy state* i.e. a state containing another state diagram. The actions and conditions of a state machine are specified in additional block diagrams or in ESDL. Additional public methods can be specified in separate diagrams. Chapter "State Machines" in the ASCET reference guide describes in detail what a state machine is and how it works.

Specifying state machines is similar to specifying block diagrams, and you can also use block diagrams as parts of a state machine. Many parts of the description of block diagram editor functionality are the same for the state machine editor. These parts are not described separately here, so before reading this chapter, you should be familiar with specifying block diagrams in the block diagram editor. chapter 4.1 on page 178 describes the block diagram editor.

Specifying a state machine consists of the following steps, which are described in subsequent sections of this chapter:

- Drawing a state diagram (i.e. arranging the states and transitions).
- Specifying the conditions and actions.
- Assigning the conditions and actions to the states and transitions in the state diagram.
- Specifying public methods in a separate diagram.

Special Menu Options

The menu options of the state machine are mostly identical to tohose in the block diagram editor (see "Description of Menu Options" on page 184). Only the **Diagram** menu contains some special menu options.

- **Diagram**
  - *Add Diagram*

    Creates a new diagram.
    (*Public* → contains only public methods,
    Actions/Conditions BDE → contains actions and conditions as block diagrams,
    Actions/Conditions ESDL → contains actions and conditions in ESDL)
  - *Add Trigger*

    Creates a trigger.
  - *Add Action*

    Creates an action (only available in the block diagram editor for status

    diagrams).
  - *Add Condition*

    Creates a condition
  - *Create State Code Comments*

    Creates a comment in the first line of ESDL code in states and transitions (cf. page 298).

4.2.1 Drawing the State Diagram

Creating a new state machine:

- In the Component Manager, select a folder for the new state machine.
- Select the menu function **Insert** → **State Machine**

or

- click on the **Insert State Machine** button.

  A new state machine is created.
- Enter a name for the state machine.
- In the menu bar, select **Component** → **Edit Item**

or

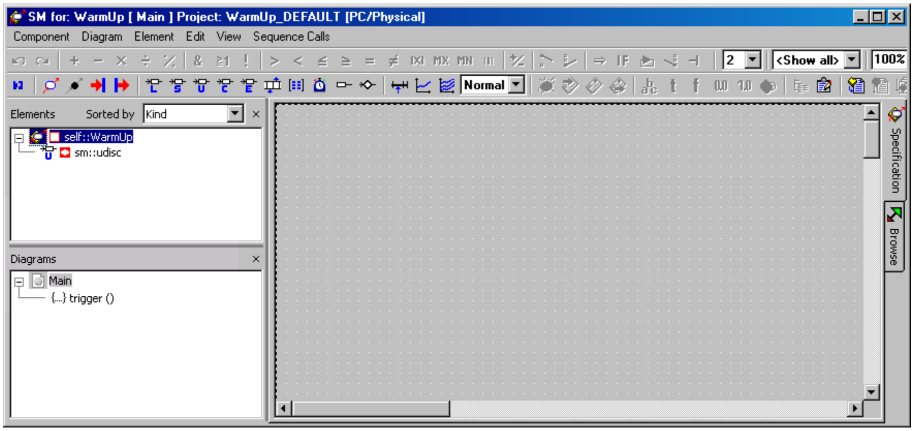
- press <RETURN>

or

- in the "Elements" list, double-click on the state machine name.

  The state machine editor opens.

  By default, the state diagram is open and the operator buttons on the toolbar are disabled. A state diagram does not contain operators.



To lay out the states for the diagram:

- Click on the **State** button to load the mouse cursor with a new state.
- Click in the drawing area where you want to position the state.

  A state symbol appears where you clicked.
- Repeat these two steps for all other states you want to lay out.

Each state machine must have a start state. The state machine is in start state when the class is initially activated.

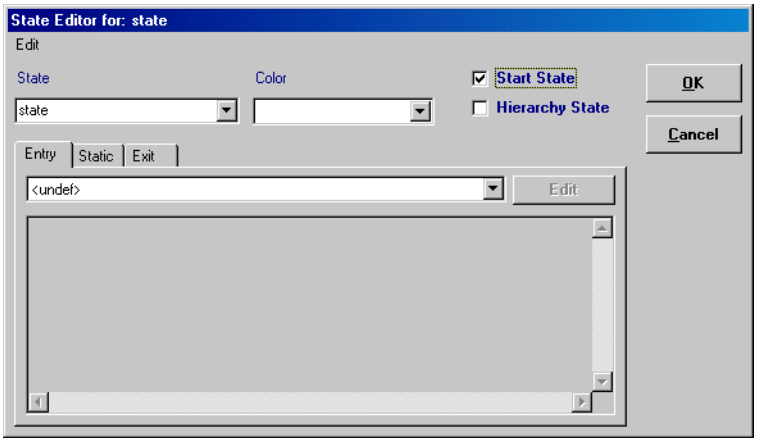
Define the starting state:

- Right-click on the state intended as the start state and select **Edit State** from the context menu.

or

- Double-click on the state symbol.

  The state editor ("State Editor" dialog box) is opened.


State Editor for: state

Edit

State: state

Color:

[✓] Start State

[ ] Hierarchy State

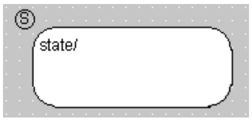
Entry | Static | Exit

<undef>

- Tick the **Start State** option.

- Click **OK**.

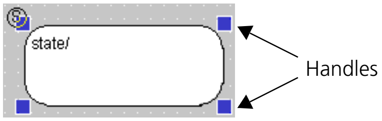
  The selected state displays a circled 'S' in the upper left corner to indicate that it is the start state.



Editing a state:

- To move state symbols, simply drag them to the position you want in the drawing area.

  When you move a symbol, the connection lines attached to it automatically follow and reroute as necessary.
- To change the size of a state symbol, click on it.



- Drag the handles until the symbol has the desired size.
- You can cut, copy and paste state symbols just like other diagram elements using the clipboard (see page 223).

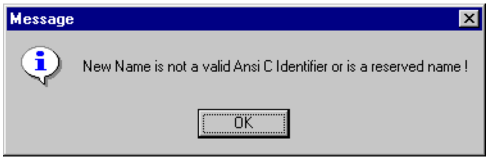
By default, the first state is named `state`, the next is `state_1`, etc. You can change the names if you like, only keep in mind that each name has to be unique and ANSI C compliant.

Renaming states:

- In the drawing area, select the state you want to rename.
- Open the state editor (page 270).
- Enter a name in the 'State' field.

- Click **OK**.

  If the name you entered is not ANSI C compliant, an error message is displayed.



Message

New Name is not a valid Ansi C Identifier or is a reserved name !

- Confirm the error message and enter a valid name.

  If you have entered an existing name, the symbol _n is attached. n is the smallest unissued number for this name.

  The new name is shown in the state symbol.

Representing states in color:

As standard, each newly added state appears in white. You can change the color as follows:

- Open the state editor for the state you want to edit.
- Select a color from the "Color" combo box.

  The state symbol is shown in the selected color.

To copy a state layout:

You can copy the layout of a state, i.e. its size and color, to another state.

- Right-click on the state whose layout you want to copy.
- From the context menu, select **State Layout** → **Copy Layout**.

  Size and color of the state are copied to the clipboard.
- Right-click on the state to which you want to copy the layout.

- From the context menu, select **State Layout** → **Paste Layout**.

  The layout in the clipboard is assigned to this state; it has the same size and layout as the first.

Besides the states, you can use the Junction diagram (see section "Junctions" in the ASCET reference guide).

Creating junctions:

- Click on the **Junction** button to load the mouse cursor with a new junction.
- Click in the drawing area where you want to position the junction.

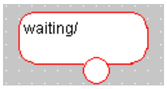
  A circle without any name appears at the point you clicked as the symbol for the junction.
- Repeat these steps, if you need more junctions.

Editing junctions:

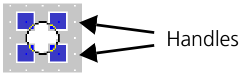
In contrast to states, you cannot assign actions to a junction. You can only move it and change the size and color.

- To move junctions, simply drag them to the position you want in the drawing area.

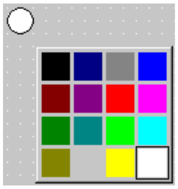
  When you move a junction, the connection lines attached to it automatically follow and reroute as necessary.
- If a junction and a state, or two junctions, overlap, the borders of both turn red. Pull one of the objects so that they are placed side by side.



- To change the size of a junction, click on it.



- Drag the handles until the symbol has the desired size.
- You can cut, copy and paste junctions just like other diagram elements using the clipboard (see page 223).
- To change the color of the junction, click on the junction with the right mouse button and select **Edit Color** from the context menu.

  A selection box opens containing the possible colors.
- Select a color.

  The junction is shown in the selected color.

To create a transition:

- Click the **Connect button**

or

- right-click in the drawing area (but not on an element).

  The connection mode is activated, the cursor changes to a crosshair.
- Click inside the state symbol or the junction where the connection is to start and drag the mouse cursor to the next state or junction.

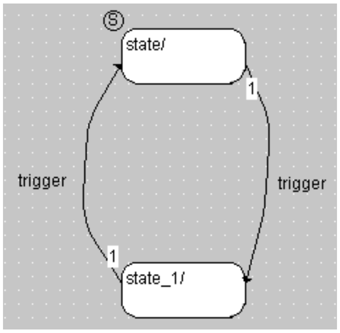
  A line is drawn after the cursor. The line starts on the edge of the symbol in which you clicked.

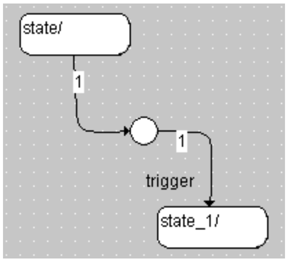
Note

It is not possible to create a transition from a junction to itself or to another junction.

- Click inside the symbol you want to connect to.

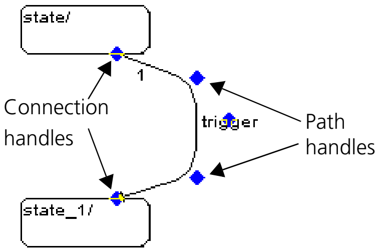
  A transition is drawn between the two symbols. It has an arrowhead pointing from the first state/junction to the second state/junction symbol.





Changing the path of a transition:

- Click on the transition to display the path and connection handles in the diagram.



- Drag the path handles to change the path of the transition.
- Drag the connection handles to change the position of the connection on the state symbol.

  You can assign an existing connection to another source or target state simply by dragging the connection handles.

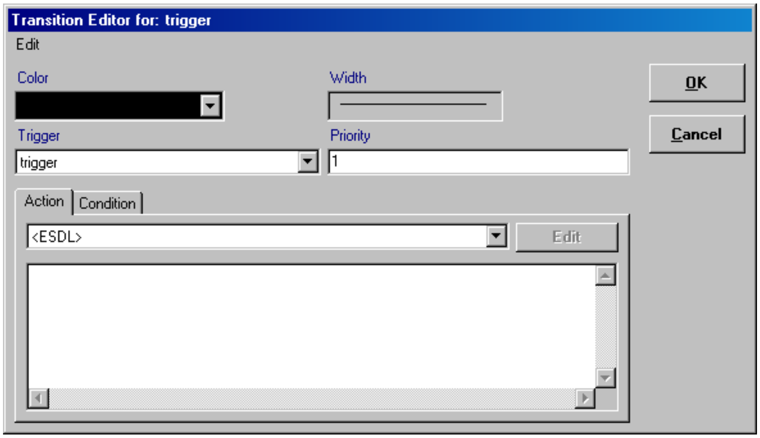
Changing the appearance of a transition:

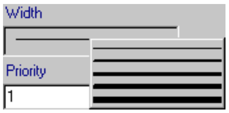
- Right-click on a transition and select **Edit Transition** from the context menu.

or

- Double-click on the transition.

  The transition editor ("Transition Editor" dialog box) opens.





- Select a color from the "Color" combo box.
- Select the line width from the "Width" combo box.
- Click **OK** to confirm your changes.

  The transition is shown with the selected color and line thickness.

One trigger is created with a new state machine by default. You can add more triggers, but keep in mind that the use of several triggers in one state machine leads to extended program code (see "Optimized for Code Size" in the ASCET reference guide).

Inserting a trigger:

- Select **Diagram** → **Add Trigger** to generate a new trigger.

  The trigger is added in the 'Diagrams' pane. Its name is highlighted to allow editing of the location and position.
- Enter a name for the trigger and press <ENTER>.

4.2.2 Hierarchy States

State diagrams can be hierarchical (see "Hierarchy" in the ASCET reference guide), i.e. a state can contain a different state diagram. When the state machine enters a hierarchy state, it starts with the start state of the subdiagram contained in the hierarchy state. If, however, the hierarchy has a history (see "History" in the ASCET reference guide), the substate activated upon a transition to the hierarchy state is the one the subdiagram was in when the hierarchy state was last left.

A distinction is made between closed and open hierarchy states. Unlike a closed hierarchy state, where the subdiagram is created on a new drawing level, the subdiagram in an open hierarchy state is on the same drawing level. However, both hierarchy types have the same functionality. This means that when you are using open hierarchy states, there is not need to switch between different drawing levels.

Open and closed hierarchy states can exist within a state diagram simultaneously; however, a single state can only have one hierarchy type. Incorrect construction from overlapping states in the drawing area is displayed by a change of color on the state symbols.

Code for a hierarchical state machine can be generated either flat (a single `switch` statement for all (basis) states) or hierarchical (nested `switch` statements according to the hierarchy); the latter can considerably reduce the code size (see section "Hierarchical Code Generation" in the ASCET reference guide).

To activate/deactivate hierarchical code generation:

to activate/deactivate hierarchical code generation, proceed as follows.

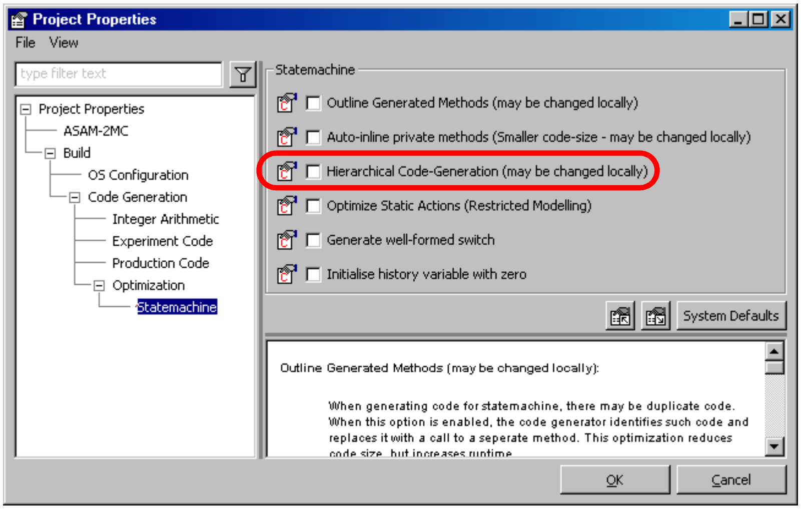
- Open the project that contains your state machine.

  This may also be the default project.

- In the project editor, open the "Project Properties" window (page 408).

1. deactivate

- In the "Statemachine" node (page 422), deactivate the **Hierarchical Code-Generation (may be changed locally)** option to deactivate hierarchical code generation for all state machines in the project.



Project Properties

File View

type filter text

Project Properties
- ASAM-2MC
- Build
  - OS Configuration
  - Code Generation
    - Integer Arithmetic
    - Experiment Code
    - Production Code
    - Optimization
      - Statemachine

Statemachine

- [ ] Outline Generated Methods (may be changed locally)
- [ ] Auto-inline private methods (Smaller code-size - may be changed locally)
- [ ] Hierarchical Code-Generation (may be changed locally)
- [ ] Optimize Static Actions (Restricted Modelling)
- [ ] Generate well-formed switch
- [ ] Initialise history variable with zero

System Defaults

Outline Generated Methods (may be changed locally):

When generating code for statemachine, there may be duplicate code. When this option is enabled, the code generator identifies such code and replaces it with a call to a seperate method. This optimization reduces code size, but increases runtime

OK Cancel

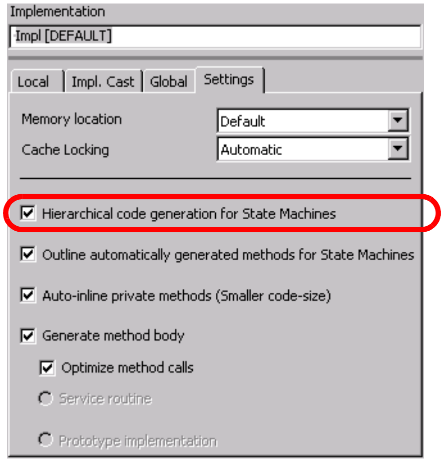
In that case, the settings of individual state machines are irrelevant.

2. activate

- Activate the **Hierarchical Code-Generation (may be changed locally)** option to activate hierarchical code generation for all state machines in the project.
- Close the "Project Properties" window.

  The global setting in the project properties alone is not sufficient to generate hierarchical code.

- Select **Component** → **Edit Implementation** to open the implementation editor of each state machine in the project for which you want to generate hierarchical code.
- In the "Settings" tab, activate the **Hierarchical code generation for State Machines** option.


Implementation

Impl [DEFAULT]

Local | Impl. Cast | Global | Settings

Memory location: Default

Cache Locking: Automatic

☑ Hierarchical code generation for State Machines

☑ Outline automatically generated methods for State Machines

☑ Auto-inline private methods (Smaller code-size)

☑ Generate method body

☑ Optimize method calls

○ Service routine

○ Prototype implementation

- Close the implementation editor.

  Hierarchical code generation takes place only if both options are activated.

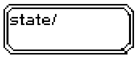
Closed Hierarchy State

To add a closed hierarchy state:

- Add the state that is to become a hierarchy state to the parent diagram.
- Right-click on the state and select **Edit State** from the context menu.
- In the state editor, activate the **Hierarchy State** option.

- Click **OK**.

  The state becomes a hierarchy state which is shown by the double-lines on the state symbol.



To move between hierarchy levels:

- Double-click on a hierarchy state

or

- right-click on the hierarchy state and select **Next Level** from the context menu.

  You move one level down, i.e. the state diagram contained inside the hierarchy state is shown. Here, you can specify the content of the hierarchy state.

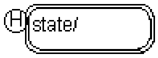
- While inside a hierarchy state, double-click in the drawing area (not on a diagram item).

  You are moved back up one level.

To set up a hierarchy state with a history:

- Right-click on the hierarchy state and select **History** from the context menu.

  The hierarchy state now has a history. When it is entered, the transition ends in that substate the hierarchy was in when it was left, even if that is not the start state of the hierarchy.



It is possible to connect states that are not in the same hierarchy with pins on hierarchical state symbols. A connection between states via a pin is the same as if the states were connected directly.

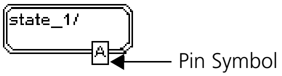
To add a pin to a hierarchy state:

- Right-click on a hierarchy state.

- Select **Add Pin** from the context menu.

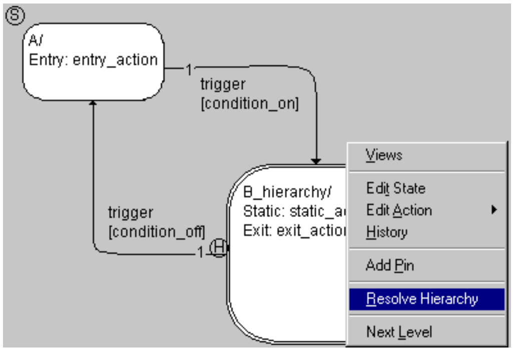
  A pin with a default name is created on the state symbol. A corresponding pin is drawn inside the hierarchy.

  You can now connect a transition to the pin on the state symbol, and then connect the pin inside the hierarchy to a state.

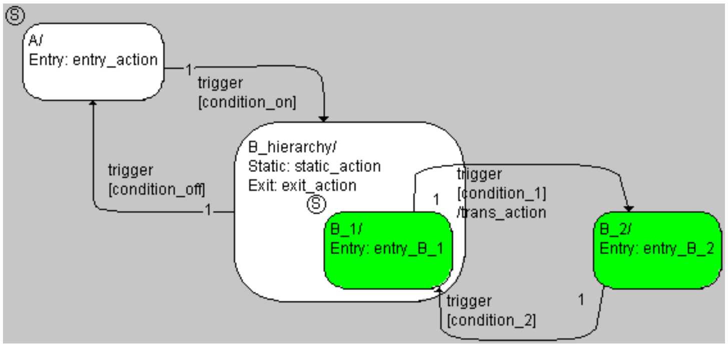


Resolving a hierarchy state:

- Right-click on a hierarchy state.
- Select **Resolve Hierarchy** from the context menu.



All the elements and their connections are copied from the hierarchy state to the higher-level drawing area in the same position. This creates a second independent state diagram with a start state. The double lines which highlighted the closed hierarchy state are deleted. If the selected state had a history, this is also deleted.



After resolving the hierarchy, the state machine is often unclear, and it (usually) does not have the same functionality as before. In the example, the states `B_1` and `B_2` were, at the beginning, contained in the close hierarchy state `B_hierarchy`. Now only `B_1` is contained in `B_hierarchy`. If, therefore, the transition from `B_1` to `B_2` (or vice-versa) occurs, other actions are also executed (see also "Semantics: Hierarchical State Machines" in the ASCET reference guide).

To create the required functionality and also to make the diagram clearer, post-editing is required.

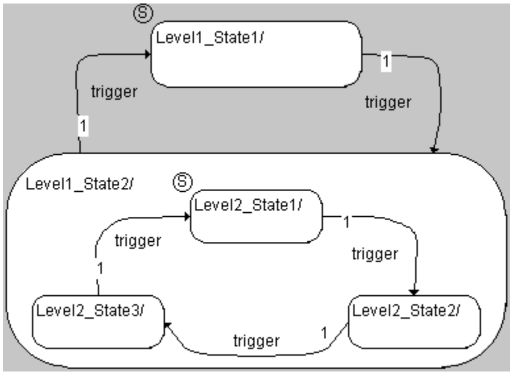
Open Hierarchy State

To add an open hierarchy state:

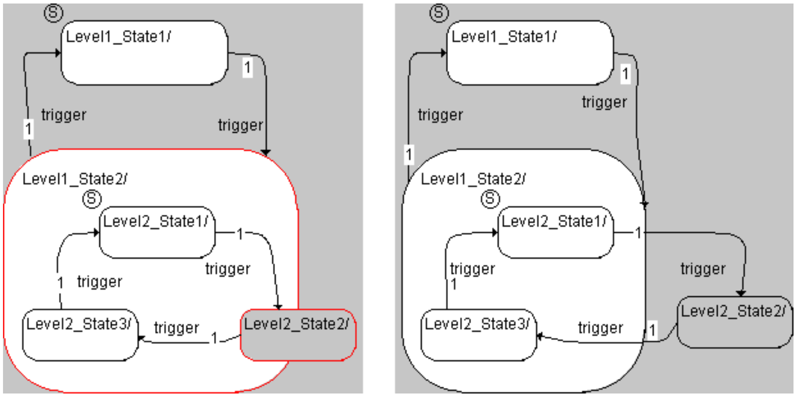
- Increase the area of the state that is to contain the hierarchy by dragging the handles.
- Add all the necessary states and transitions for the subdiagram to the state symbol in the selected hierarchy state.

- Select a state intended as the start state for the subdiagram.

  The method used to create a state diagram within a state symbol is identical to creating a state diagram in the drawing area.



Transitions from a substate to a state outside the hierarchy are possible. Keep in mind, however, that different actions are performed upon a transition between different hierarchy levels than upon a transition within the same hierarchy level (see Example 12 and Example 15 in the in the ASCET reference guide). A state is considered outside a hierarchy when it is placed fully or partly outside the hierarchy state (state `Level2_State2` in the figure).



As with closed hierarchy states, the user can set up a hierarchy state with a history.

4.2.3 Specifying Conditions and Actions

Every state can have an entry, a static and an exit action, every transition can have a trigger, a condition and a transition action attached to it. Conditions (see "Conditions" in the ASCET reference guide) and actions (see "Actions" in the ASCET reference guide) are similar to methods, and they are specified the same way as the methods of classes.

A condition is tested each time the state machine is in a state that has a transition with this condition attached to it. If the condition is true, a transition takes place. Therefore a condition always has `true` or `false` as its return value. Entry, exit and transition actions - if present - are carried out each time a transition takes place, static actions are carried out when no transition takes place. Chapter 2.5.2 til chapter 2.5.6 in the ASCET reference guide describe the functions of a state machine in detail.

You can specify conditions and actions either in separate diagrams (`Action-Condition` diagrams) in the form of block diagrams or ESDL code. In that case, the state machine then contains at least two diagrams. Alternatively, you can specify actions or conditions in ESDL directly in the state or transition editor; in this case no separate diagram is required.

Static actions of hierarchy states can be optimized regarding code size; see chapter 2.5.8, section "Static actions of hierarchy states", in the ASCET reference guide.

To activate/deactivate optimization of static actions:

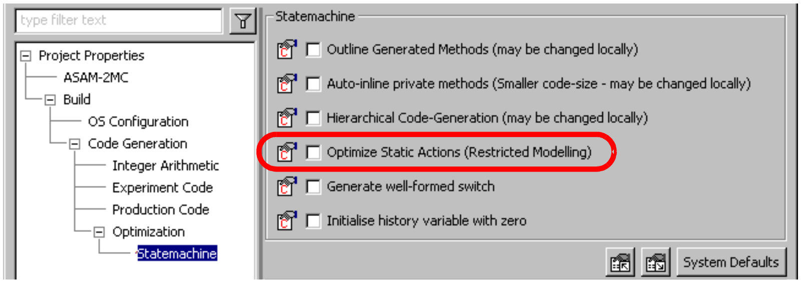
to activate/deactivate optimization of static actions in hierarchy states, proceed as follows.

- Open the project that contains your state machine.

  This may also be the default project.
- In the project editor, open the "Project Properties" window (page 408).

1. deactivate

- In the "Statemachine" node (page 422), deactivate the **Hierarchical Code-Generation (may be changed locally)** option to deactivate hierarchical code generation for all state machines in the project.


type filter text

- Project Properties
  - ASAM-2MC
  - Build
    - OS Configuration
    - Code Generation
      - Integer Arithmetic
      - Experiment Code
      - Production Code
      - Optimization
        - Statemachine

Statemachine

- [ ] Outline Generated Methods (may be changed locally)
- [ ] Auto-inline private methods (Smaller code-size - may be changed locally)
- [ ] Hierarchical Code-Generation (may be changed locally)
- [ ] Optimize Static Actions (Restricted Modelling)
- [ ] Generate well-formed switch
- [ ] Initialise history variable with zero

2. activate

Note

When you activate the optimization, modeling is restricted as follows: If a substate of the state machine contains a direct transition out of it's hierarchy state, this transition must have the highest priority of all transitions from that substate.

- Activate the **Optimize Static Action (Restricted Modeling)** option to activate the optimization for all state machines in the project.

  The optimization of static actions takes place. The changes in code generation thus introduced can alter the behavior of the state machine. If you activate the option for an existing state machine, check it's behavior carefully.

Conditions and Actions in Separate Diagrams

You can specify actions and conditions as block diagrams or ESDL code in separate `ActionCondition` diagrams. A state machine can contain any number of these `ActionCondition` diagrams; an individuakl diagram contains either block diagrams or ESDL code.

First, you create the required diagrams. After that, actions and conditions can be specified like normal block diagrams or ESDL code. Specifying block diagrams is explained in "Creating Block Diagrams" on page 206, specifying ESDL code is explained in "The ESDL Editor" on page 328.

Generating a diagram for actions/conditions:

1. Block Diagram

- Select **Diagram** → **Add Diagram** → **Actions/Conditions BDE**

or

- right-click in the "Diagrams" pane and select **Add Diagram** → **Actions/Conditions BDE** from the context menu.

  The diagram `ActionCondition_BDE` is created.

- Enter a name for the diagram in the "Diagrams" pane.

2. ESDL

- Select **Diagram** → **Add Diagram** → **Actions/Conditions ESDL**

or

- right-click in the "Diagrams" pane and select **Add Diagram** → **Actions/Conditions ESDL** from the context menu.

  The diagram `ActionCondition_ESDL` is created.

- Enter a name for the diagram in the "Diagrams" pane.

Adding a condition:

- In the "Diagrams" pane, select the name of the diagram to which you want to add the condition.
- Select **Diagram** → **Add Condition**

or

- right-click in the "Diagrams" pane and select **Add Condition** from the context menu.

  A new condition is created and its name is shown in the "Diagrams" pane.

  A condition has a logical return value. The return value for the condition is created automatically. When you open the diagram that contains the condition, the return value is displayed in the "Elements" pane.
- Enter a name for the condition and press <ENTER>.

An action has, as standard, no arguments and no return value. Therefore, no interface elements are created for an action, but other than that, adding actions works analogous to adding conditions, using the menu **Diagram** → **Add Action** or the context menu **Add** → **Action**.

Opening a diagram for actions/conditions:

- In the "Diagrams" pane, select the `Action-Condition` diagram you want to open.
- Double-click on the diagram

or

- select **Diagram** → **Load Diagram**.

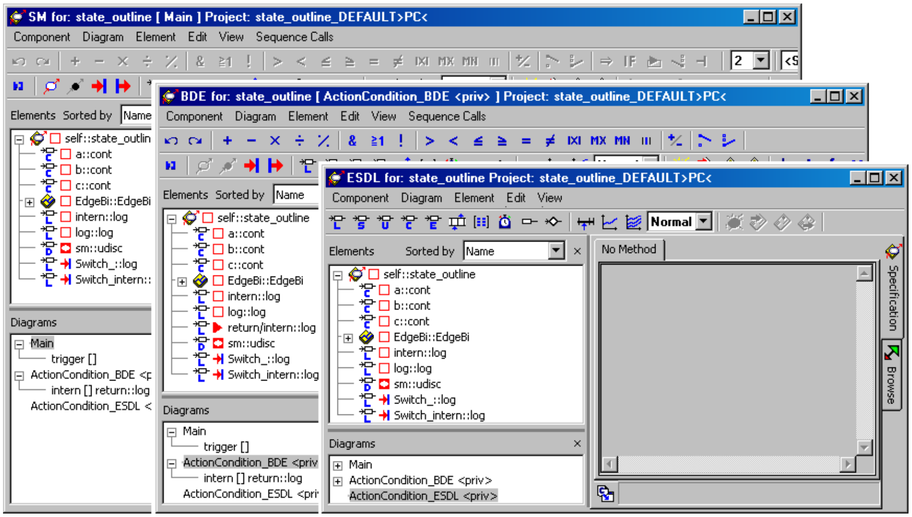
  If the currently loaded diagram contains unsaved changed, you are asked whether you want to save the changes.

  – Click **Yes** to save the changes

  or

– Click **No** to reject the changes

The `ActionCondition` diagram opens in a new editor window.



Specify the conditions and actions as for all the other block diagrams or ESDL components, see chapters 4.1 "The Block Diagram Editor" and 4.4 "The ESDL Editor". Only the menu options and buttons for code generation are disabled; these are available only in the state machine editor.

Code for actions/conditions thus specified is generated either as separate functions or inserted on the spot during code generation (*auto-inlining*; see "Optimizing the State Machine" in the ASCET reference guide).

To activate/deactivate auto-inlining:

to activate/deactivate auto-inlining, proceed as follows.

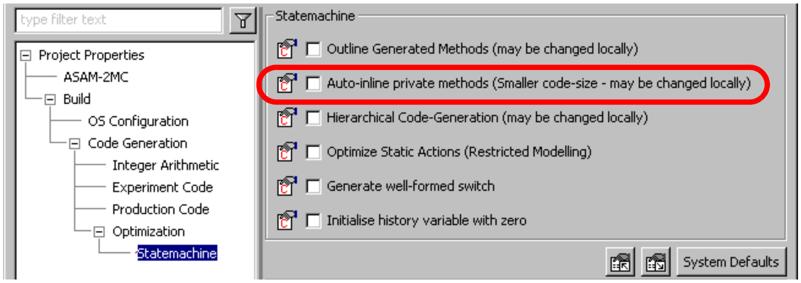
- Open the project that contains your state machine.

  This may also be the default project.

- In the project editor, open the "Project Properties" window (page 408).

1. deactivate

- In the "Statemachine" node (page 422), deactivate the **Auto-inline private methods (Smaller code size - may be changed locally)** option to deactivate auto-inlining for all state machines in the project.



Statemachine

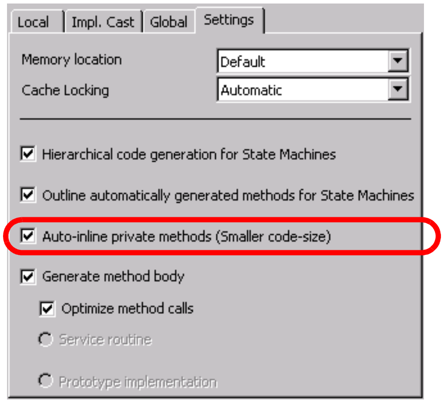
- [ ] Outline Generated Methods (may be changed locally)
- [ ] Auto-inline private methods (Smaller code-size - may be changed locally)
- [ ] Hierarchical Code-Generation (may be changed locally)
- [ ] Optimize Static Actions (Restricted Modelling)
- [ ] Generate well-formed switch
- [ ] Initialise history variable with zero

In that case, the settings of individual state machines are irrelevant.

2. activate

- Activate the **Auto-inline private methods (Smaller code size - may be changed locally)** option to activate auto-inlining for all state machines in the project.
- Close the "Project Properties" window.

  The global setting in the project properties alone is not sufficient to activate auto-inlining.
- Select **Component** → **Edit Implementation** to open the implementation editor of each state machine in the project for which you want to activate auto-inlining.
- In the "Settings" tab, activate the **Auto-inline private methods (Smaller code size)** option.


Local | Impl. Cast | Global | Settings

Memory location: Default

Cache Locking: Automatic

☑ Hierarchical code generation for State Machines

☑ Outline automatically generated methods for State Machines

☑ Auto-inline private methods (Smaller code-size)

☑ Generate method body

☑ Optimize method calls

○ Service routine

○ Prototype implementation

- Close the implementation editor.

  Auto-Inlining takes place only if both options are activated.

  You can use this option to exclude individual state machines from auto-inlining.

Using Conditions and Actions

You must explicitly assign actions and conditions from an `ActionCondition` diagram to the transitions or states in which they are used.

Assigning triggers, priorities, conditions and actions to a transition:

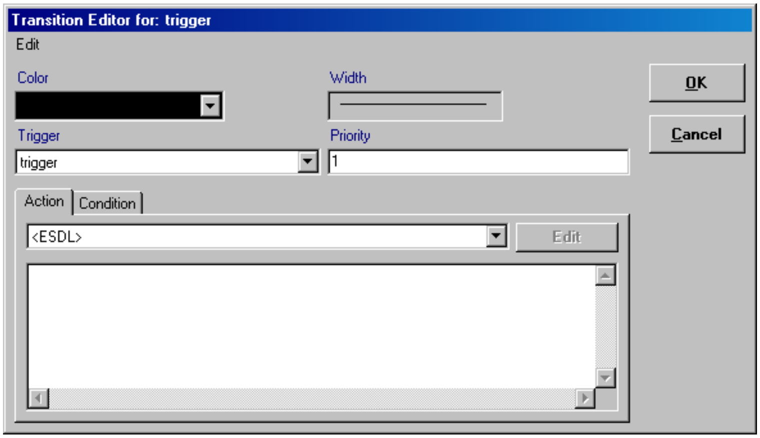
Every transition has to have a trigger and a priority, but the condition and the transition action are optional.

- Double-click on the transition or the transition segment

or

- right-click on the transition/segment and select **Edit Transition** from the context menu.

  The transition editor appears.



Trigger: trigger

Priority: 1

<ESDL>

- Enter a number in the "Priority" field to assign a priority to the transition.

Note

The higher the number selected, the higher the priority of the transition.

If there is more than one transition/segment with the same trigger leading from a state or junction, their conditions are checked in the order of priority. Each transition leading from a state/junction with the same trigger has to have a unique priority.

- Select a trigger from the "Trigger" combo box.

Note

If a transition is split into two transition segments by a junction, only one of the segments may possess a trigger.

All trigger events defined in the state diagram are listed in this box.

- Select a transition action from the combo box on the "Action" tab.

Note

A transition segment leading into a junction may not possess an action. The combo box is deactivated.

All the actions specified in an `ActionCondition` diagram are listed in the combo box and can be assigned as transition actions.

- Select a condition from the combo box on the "Condition" tab.

All conditions specified in an `ActionCondition` diagram are listed in this box.

Note

If no condition is specified, it means that the condition is always true, i.e. the transition always takes place.

- Click **OK**.

The trigger event name, the priority and the names of the condition and action of the transition/segments are shown in the diagram.

Each state can have an entry action, a static action and an exit action. All these are optional.

Assigning actions to a state:

- Double-click on a state

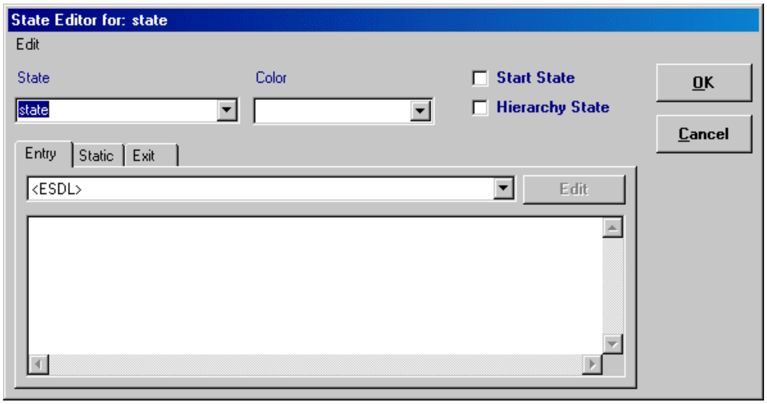
or

- right-click on the state and select **Edit State** from the context menu

or

- select **Edit Action** → ***<action>*** from the context menu to edit a particular action.

  The state editor opens.

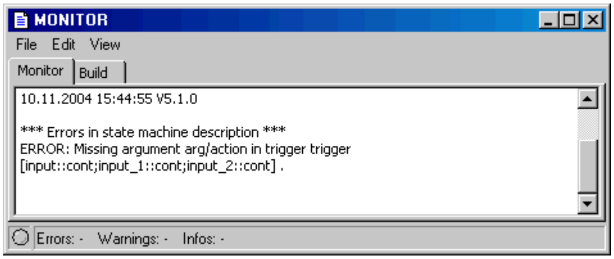


- On the "Entry" tab, select an entry action from the combo box.
- On the "Static" and "Exit" tabs, select the static and exit actions.

  You can use all the actions here.
- Click **OK**.

  The names of the actions are displayed in the state symbol.

If you have assigned actions or conditions wirh arguments, a test is carried out when you leave the state or transition editor to see if all the relevant triggers have the corresponding arguments with identical names and types. If this is not the case, an error message appears in the ASCET monitor window.



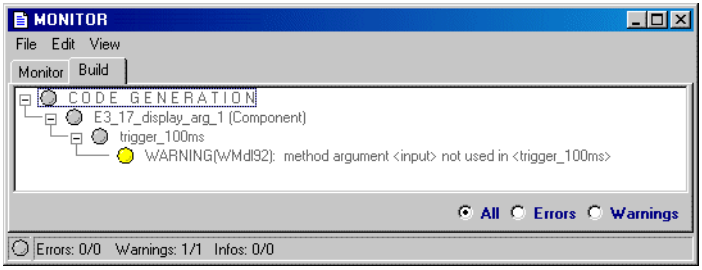
```
10.11.2004 15:44:55 V5.1.0

*** Errors in state machine description ***
ERROR: Missing argument arg/action in trigger trigger
[input::cont;input_1::cont;input_2::cont] .
```

Errors: - Warnings: - Infos: -

Generate the trigger elements specified in the error message as described on page 301.

If you have generated a trigger argument in reverse, and it is not used in any action or condition, this has no effect on the function of the state machine. On code generation, a warning is only displayed in the "ASCET Errors/Warnings" window.

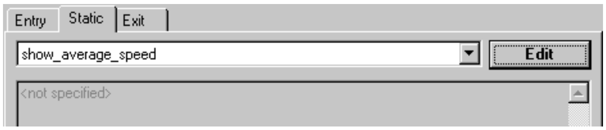


```
CODE GENERATION
  E3_17_display_arg_1 (Component)
    trigger_100ms
      WARNING(WMdl92): method argument <input> not used in <trigger_100ms>
```

[✓ All] Errors Warnings

Errors: 0/0 Warnings: 1/1 Infos: 0/0

Once you have assigned actions or conditions from an **ActionCondition** diagram, the **Edit** button on the various tabs becomes active. You can use it to edit the action/condition assigned on the respective tab.



To edit actions/conditions:

- Open the state editor or transition editor.

- In one of the tabs, assign an action or condition.
- To edit the action or condition, click **Edit**.
- In the "Confirm" window, confirm the saving of your input to the state or transition editor.

  The state/transition editor is closed, the settings are adopted.

  Another "Confirm" window appears.
- In the new window, confirm the save of your inputs to the state diagram.

  The state diagram is saved.

  The diagram containing the assigned action or condition open in a separate editor window.

Conditions and Actions in the State Diagram

It is possible to type the ESDL code for a condition or action in the state diagram, directly into the state editor or transition editor. In this case it is not necessary to create a specification in a separate diagram, the **Edit** button is inactive.

You can declare variables and parameters by clicking on an element button in the state machine editor and typing a name into the "Elements" pane. You can then use those variables in the ESDL code by typing in their names. Likewise, you can refer to any trigger arguments, imported components, their methods or the elements defined within them by typing the name. This makes it possible to specify complex state machine behavior with just a few lines of code, and without any additional diagrams. The ESDL language is described in chapter 5 in the ASCET reference guide.

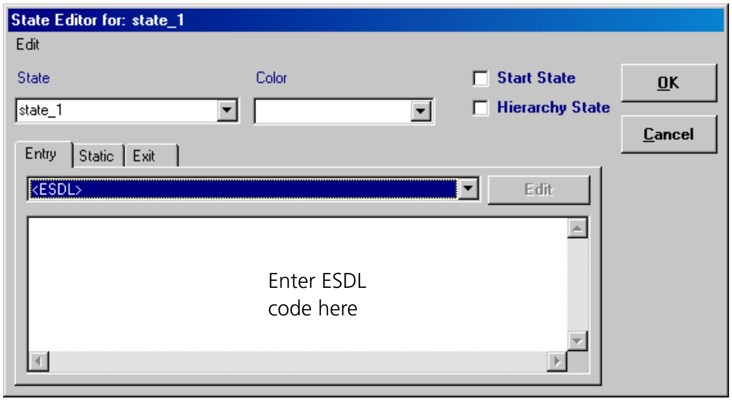
Specifying actions in the state editor:

- Right-click on the state whose actions you want to specify.
- Select **Edit Action** → **Entry** from the context menu to edit the entry action.
- Select **Edit Action** → **Static** from the context menu to edit the static action.
- Select **Edit Action** → **Exit** from the context menu to edit the exit action.

  The state editor opens in the requested tab.

- Select **<ESDL>** from the combo box.

  The input field below the combo box is activated.



- Type the ESDL code into the input field.

  You can undo the last input with <CTRL> + <Z>.

- Click **OK** to accept your input.

The **Edit** menu in the state editor provides several functions to ease the specification of actions in ESDL.

To cut, copy and paste ESDL code:

- Highlight the section of code you want to edit.

or

- Select **Edit** → **Select All** to highlight all the code of the current action.
- Select **Edit** → **Cut** to remove the selected code and store it on the clipboard.
- Select **Edit** → **Copy** to copy the highlighted code to the clipboard.
- Select **Edit** → **Paste** to paste the code from the clipboard to any tab.

You can search and replace code, see section "To search and replace C Code:".

You can also insert code from an external source into the action, or save the code you specified in an external file, see section "Integrating an external source file:".

You can print the specified code, see section "Printing the C code:".

The transition editor, which is used to specify conditions and transition actions, has the same functionalities as the state editor.

Specifying conditions and transition actions the transition editor:

- Right-click on the transition.
- Select **Edit Transition** from the context menu.

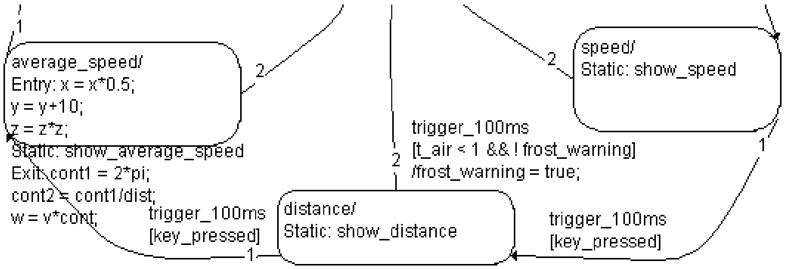
or

- Double-click on the transition.

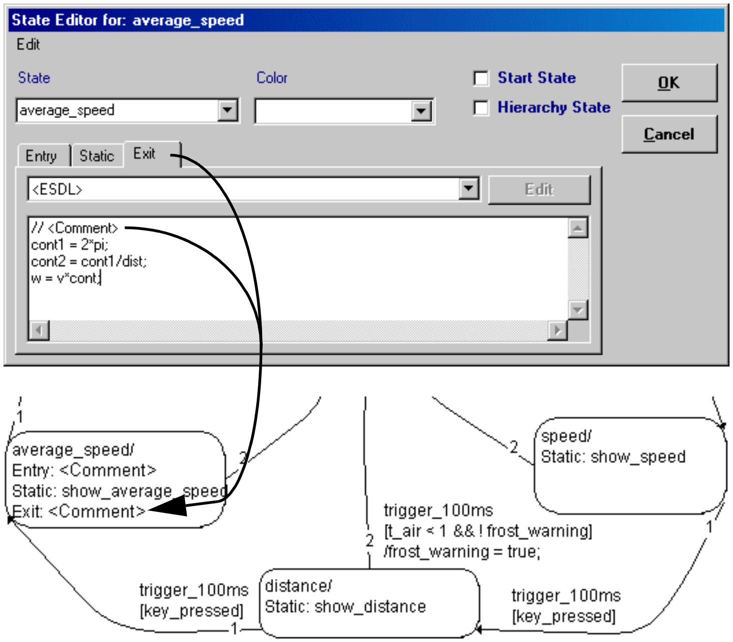
  The transition editor opens.
- On the "Condition" or "Action" tab, select <ESDL> from the combo box.

  The input field below the combo box is activated.
- Type the ESDL code into the input field.
- Click **OK**.

The ESDL code of all conditions and actions specified at a state or transition is displayed in the state diagram. For complicated state machines, it can thus quickly become rather crowded.



You can avoid that by entering a comment in the first line when you add conditions or actions. One-line comments are marked by two leading slashes `//`, comments of any length are included in `/* <Comment> */. In this case, only the comments are displayed in the diagram.`



```
// <Comment>
cont1 = 2*pi;
cont2 = cont1/dist;
w = v*cont;
```

You can also add the comment lines automatically at a later time.

Automatic insertion of comment lines:

- Open the state machine in the state machine editor.
- Select **Diagram** → **Create State Code Comments**.

  In each condition and action specified in the state diagram, a comment with the text of the first code line is added as first line.

When code is generated for a state machine, parts of actions and conditions specified at the state or transition are generated as separate methods (*outlining*). Thre prerequisites for outlining are desctibed in chapter "Optimizing the State Machine" in the ASCET reference guide. You can disable outlining.

To activate/deactivate outlining of actions/conditions:

to activate/deactivate outlining, proceed as follows.

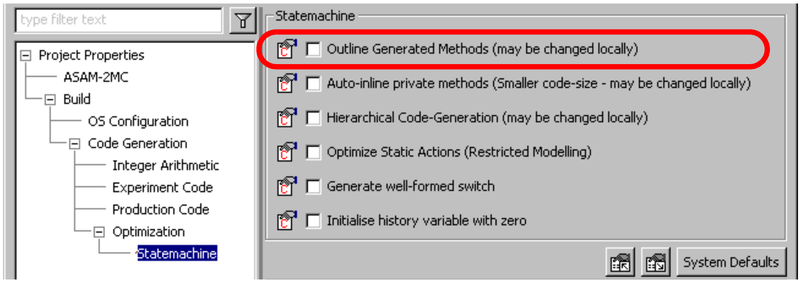
- Open the project that contains your state machine.

  This may also be the default project.

- In the project editor, open the "Project Properties" window (page 408).

1. deactivate

- In the "Statemachine" node (page 422), deactivate the **Outline Generated Methods (may be changed locally)** option to deactivate outlining for all state machines in the project.



Statemachine

☐ Outline Generated Methods (may be changed locally)

☐ Auto-inline private methods (Smaller code-size - may be changed locally)

☐ Hierarchical Code-Generation (may be changed locally)

☐ Optimize Static Actions (Restricted Modelling)

☐ Generate well-formed switch

☐ Initialise history variable with zero

In that case, the settings of individual state machines are irrelevant.

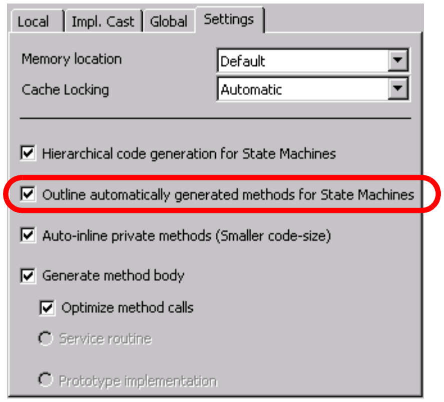
2. activate

- Activate the **Outline Generated Methods (may be changed locally)** option to activate outlining for all state machines in the project.
- Close the "Project Properties" window.

  The global setting in the project properties alone is not sufficient to activate outlining.

- Select **Component** → **Edit Implementation** to open the implementation editor of each state machine in the project for which you want to activate outlining.

- In the "Settings" tab, activate the **Outline automatically generated methods for State Machines** option.


Local | Impl. Cast | Global | Settings

Memory location: Default

Cache Locking: Automatic

- [✓] Hierarchical code generation for State Machines
- [✓] Outline automatically generated methods for State Machines
- [✓] Auto-inline private methods (Smaller code-size)
- [✓] Generate method body
  - [✓] Optimize method calls
  - ○ Service routine
  - ○ Prototype implementation

- Close the implementation editor.

  Outlining takes place only if both options are activated.

  If required, code for the actions/conditions is generated as separate methods during code generation.

Communication with Other Components

A state machine can communicate with other ASCET components. For this, there are several options: inputs and outputs, trigger arguments and public methods (see also section "State Machines as Classes" in the ASCET reference guide).

Adding inputs and outputs to the state machine:

- Click on the **Input** or **Output** button.
- In a block diagram editor for `ActionCondi-tion` diagrams, click inside the drawing area where you want to position the input or output.

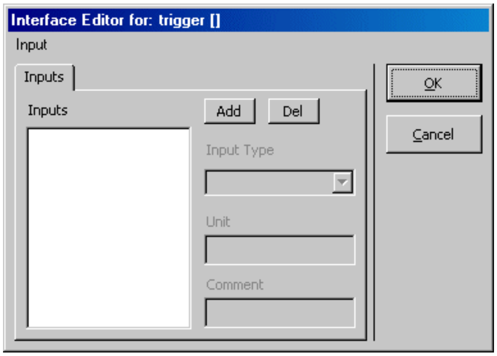
  The input or output is created. You can use the element the same way as an argument or a variable.

If you want carry out external communication using trigger arguments, a series of steps are necessary.

Firstly, you need one or more trigger arguments. The rules on which triggers must to be added to arguments and in which cases are shown on "State Machines as Classes" (page 85) in the ASCET reference guide.

Adding a trigger argument:

- Open the interface editor for the required trigger (see page 192).



Different from the interfaces of normal methods, there is only one tab here, "Inputs", and one menu, **Input**.

- Add the necessary arguments in the "Inputs" tab (see page 193).

The trigger argument belongs to a public method in the same diagram as the state diagram. Therefore, you can use it exactly as you would use variables or parameters, if you specify conditions or actions in the transition or state (see "Conditions and Actions in the State Diagram" on page 295).

If you want to use the trigger argument in an action or condition specified in an `ActionCondition` diagram, you must generate an argument of the same name and the same type in the action or condition. The arguments in triggers and actions/conditions are mapped according to their name and their type.

Adding arguments to conditions/actions:

- In the "Diagrams" pane, select the required action or condition.

- Open the interface editor.

  The interface editor for actions/conditions is the same as for normal methods.

- In the "Arguments" tab, add the necessary arguments. Make sure that the names and the types are the same.

  With this generated argument you can create connections as with a parameter.

- If necessary, you can enter a comment for the return value of a condition in the "Return" tab.

  Actions can have return values; however, those values are ignored during usage as action.

4.2.4 Public Methods

You have the option of specifying public methods in a separate diagram. You can open these methods from within the state machine as well as from other components.

To create a public diagram:

- Select **Diagram** → **Add Diagram** → **Public**

or

- Right-click in "Diagrams" pane and select **Add Diagram** → **Public** from the context menu.

  A public diagram is created.

- Enter a name for the diagram in the "Diagrams" pane.

- In the "Diagrams" pane, double-click on the name of the diagram.

  The diagram opens in a separate editor window.

To add a public method:

- In the "Diagrams" pane, select a public diagram.

- Select **Diagram** → **Add Method**

or

- right-click in the "Diagrams" pane and select **Add Method** from the context menu.

  A new method is created and its name is shown in the "Diagrams" pane.

- Enter a name for the method and press <ENTER>.

The methods are specified as described in chapter 4.1 "The Block Diagram Editor". Some examples are listed in the following, but the list is by no means complete.

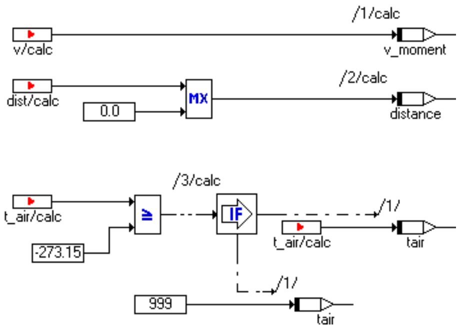
Using a public method for external communication:

- Create a public method in a public diagram.
- Add the required arguments as described on page 193.

  The arguments can be read only within the method. To be available in the state machine, their values have to be assigned to variables.

- Add the respective number of variables.
- Connect each argument with a variable.

  The variables can be used in the entire state machine. The figure lists examples to process or check the input values in the method.

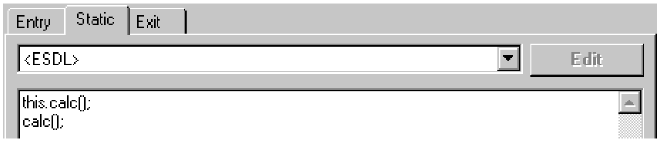


To use public methods in actions and conditions:

A public method with no return value is available in the combo box on the action tabs in the state or transition editor. A public method with a return value is available in the combo box on the "Condition" tab in the transition editor.

- Assign the public method to an action as described on page 293.
- Assign the public method to a condition as described on page 290.

If you specify actions or conditions in ESDL, you can call the public methods of the state machine exactly like the methods of imported classes. The class name is either replaced by `this` or `self`, or left out completely.



```
this.calc();
calc();
```

Both ways to specify the calling of the public method `count` in the above figure obtain the same results.

4.2.5 Experimenting with state Machines

You can experiment with a state machine in the same way as with any other component. To start the offline experimentation environment for a state machine, perform the same steps as outlined in the section "Analyzing Components" on page 258.

To analyze a diagram:

You can analyze the state diagram or one of the other diagrams of the state machine.

- Load the diagram you want to analyze into the drawing area.

- Select **Diagram** → **Analyze Diagram** to analyze the current diagram.

  The diagram is checked for syntactic errors. Errors are reported in the ASCET monitor window (see chapter 2.4 "The ASCET Monitor Window").

Note

*ESDL code in conditions or actions is **not** analyzed.*

- Click on an error message in the monitor window, "Build" tab, to have the error highlighted automatically in the state machine editor.

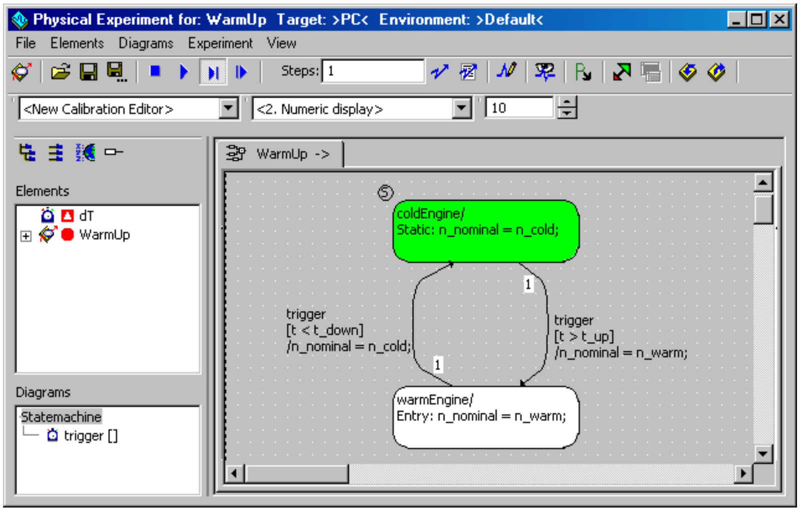
Setting up the experimentation environment also works in the same way for all kinds of components and is described in "The Experimentation Environment" on page 561. The state machine editor offers state machine animation as an additional feature.

Using the state machine animation feature:

- Start the offline experiment for the state machine.
- Set up the experiment as required.
- Switch to the state diagram.
- Right-click on a state.
- Select **Animate States** from the context menu.

- Start the simulation.

  The simulation runs as normal, but the current state is highlighted in the state diagram in the animation color.



Changing the animation color:

If you want to change the animation color, proceed as follows.

- In the Component Manager , select **Tools** → **Options**.
- In the "Options" window, select the "Fonts & Graphics" tab.
- From the "Color of Animated States" combo box, select the animation color you want.
- Click **OK**.

  The animated states are displayed with the selected color. You can change the animation color while the experiment is running.

You can look at individual states or transitions while the experiment is running.

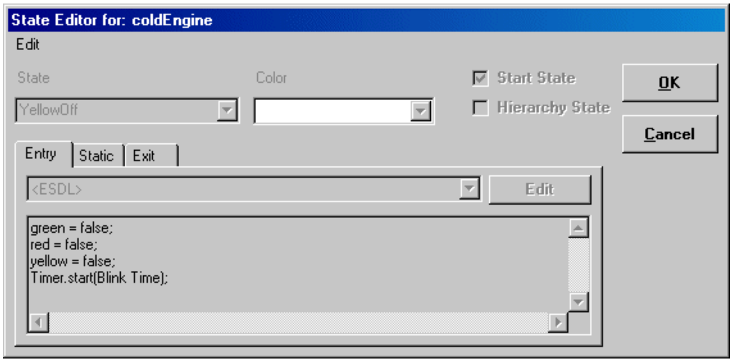
Displaying information about states or transitions:

- Right-click on a state and select **State Info** from the context menu

or

- right-click on a transition and select **Transition Info** from the context menu.

  The state editor, or transition editor, opens. All editing facilities are disabled.



```
green = false;
red = false;
yellow = false;
Timer.start(Blink Time);
```

You can look at the actions/conditions assigned, or specified in ESDL, in the tabs.

- Click **OK** or **Cancel** to close the window.

You can reset the state machine to its original state while the experiment is running.

Resetting the state machine:

- Right-click on one of the states.
- Select **Reset Statemachine** from the context menu.

  The state machine is reset to the start state.

  This does *not* affect the values of any variables within the state machine. If the experiment has not been stopped, it continues normally from the start state.

4.3 The C Code Editor

The C code editor is used to specify classes, modules, or continuous time blocks in the C programming language. Components can be specified in C code either by typing in the code for the class, or by including existing programs and modifying them as required.

This section describes how to use the C code editor for specifying classes and modules. The C code editor is based on the same principles as the block diagram editor For more information see "The Block Diagram Editor" on page 178.

Creating a C Code component:

- In the Component Manager, select a folder for the new component.
- Select **Insert** → **Class** → **C Code** or **Module** → **C Code**.

Or

- use the **Insert Class - <Type>** or **Insert Module - <Type>** buttons.
  - Use one of the arrow buttons to select the type **C Code**.
  - Click on **Insert Class - C Code** or **Insert Module - C Code**.

  A new component is created.
- Type in a name for the component and press <ENTER>.
- On the menu bar, select **Component** → **Edit Item**

or

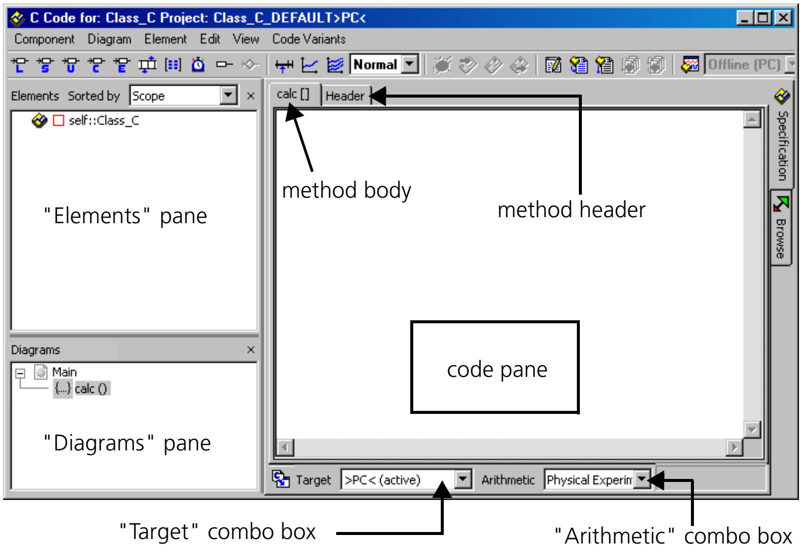
- press <ENTER>

or

- in the "1 Database" list, double-click on the component name.

  The C code editor for the new component opens.

The following illustration shows the C code editor with the most important parts labelled.



The "Elements" and "Diagrams" panes work in the same way as in the block diagram editor. The diagram feature is available, but it only serves to structure the methods of the component into groups of public and private methods. The code for the body of each method or process of a C code component is shown in the code pane, when the method or process is selected in the "Diagrams" pane. The title bar shows the name of the component, the current project and the current target.

The "Target" combo box shows the current target. A C code component can contain code for several targets, but the code has to be specified for each target individually. This is because C code is platform-dependent; you cannot assume that C code developed for one platform will run on another platform without modifications.

Different experimentation arithmetic, e.g. floating point or fixed point is possible for each target; the C code also has to be specified separately for each of them. It is possible to copy the code for each target/arithmetic combination to any other, so you only need to enter the code once, and it can then be adapted as required.

You will specify the body of a method or process in the code pane. A method corresponds to a function in C, i.e. it can have several arguments but only one return value. A process has neither arguments nor return values. The declaration of the function is generated by ASCET; the user has to define only the content of the function.

```
calc (a,b)
  {
     <user-defined content in C Code >
   }
```

The arguments and return values of the function are specified as in the Block Diagram Editor.

If you want to include any system libraries, you can do this with an ordinary include statement. This is written in the header of the class or module. Click on the "Header" tab of the "Code" pane to display the header. Variables local to a method are also declared here with normal C-statements. The scope of a variable is the method or process it has been declared in, i.e. you cannot reference a variable that was declared in one method or process in another one.

Basic elements are created as in the block diagram editor by clicking on one of the ***element*** buttons. Elements work like elements in components described by block diagrams, i.e. they can be used in all methods or processes of the component, and can also be made global. Once you have created an element, you can reference it by simply typing its name in the "Code" pane.

Similarly, you can define arguments and return values for classes, and messages for modules, as you would in a block diagram component. However, if the component is to reference other components there is a difference in syntax; you should therefore decide early on whether the component references others.

Description of the Menu Options

- **Component**
  - *Clean code generation directory*

    Deletes all files in the code generation directory.
  - *Touch*

    Forced regeneration during the next code generation
    (*Flat* → the edited components,
    Recursive → the referenced components also).
  - *Generate Code*

    Generates the code for a component
  - *Open Experiment*

    Starts an experiment.

- *Default Project*

  Allows editing the default project used for experiments w. components
  (*Edit* → Opens an editor for the default project,
  Resolve Globals → automatically creates a global element for each imported element in the component for which there is no exported element,
  Delete Unused Globals → deletes unused global elements).
- *Edit Layout*

  Opens the layout editor.
- *Edit Data*

  Opens the data editor for a component. Search of component data is possible.
- *Edit Implementation*

  Opens the implementation editor Search of component implementations is possible.
- *Edit Notes*

  Opens the Notes editor—you can make notes about the component here.
- *Check Dependency*

  Checks whether the allocation of formal parameters to the model parameters is correct.
- *Export Data*

  Exports a component data set.
- *Show Path*

  Shows the path of an included component.
- *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard
  (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path
  Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

- *File out Generated Code*

  Saves the code generated in the file system
  (*Flat* → the edited component,
  Recursive → the referenced components also)

- *View Generated Code*

  Generates the code for the component and displays it in a text editor.
  The text editor can be selected in the ASCET options window, "ASCII Editor" node (cf. page 60).

- *File Out Generic Code For External Make*

  Saves the code generated in any directory for processing at a later point using external tools, e.g. an external Make/Build process.

- *Exit*

  Closes the C code editor.

• **Diagram**

Note

This menu is also available as context menu in the "Diagrams" pane.

- *Analyze Diagram*

  Analyzes the current diagram (deactivated in the C code editor).

- *Add Diagram*

  Creates a new diagram.
  (*Public* → contains only public methods,
  Private → contains only private methods)

- *Rename Diagram*

  Renames a diagram

- *Delete Diagram*

  Deletes a diagram from the component.

- *Add Method*

  Creates a method. Available for classes and modules.

- *Add Process*

  Creates a process.

- *Add Trigger / Action / Condition*

  Deactivated; only available in the state machine editor.

- *Edit*

  Editing the method/process interface.
- *Rename*

  Renames a method/process.
- *Delete*

  Deletes a method/process.
- *Move Up*

  Moves a diagram or a method/process (upwards).
- *Move Down*

  Moves a diagram or a method/process (downwards).
- *Move*
  Moves methods/processes between diagrams.
- *Edit Implementation*

  Specifies the implementation of a method / process.

• **Element**

Note

*The **Add Item**, **Rename**, **Delete** and **Edit** functions are only available when ASCET-MD is installed.*
This menu is also available as context menu in the "Elements" pane.

- *View → Collapse all* used to collapse the tree structure in the "Elements" pane so that only the component is shown.
- *View → Expand all* expands the tree structure so that the entire content of the component is visible.
- *Add Item*

  Accepts a component as a complex element.
- *Rename (*<F2>*)*

  Renames a diagram element.
- *Delete (*<DEL>*)*

  Deletes a diagram element.

– *Show Occurrences*

  Displays all occurrences of an element (only available for block diagrams).

– *Edit*

  Editing the configuration of an element.

– *Edit Data*

  Editing the data of an element.

– *Edit Implementation*

  Editing the implementation of an element.

– *Set Cache Locking*

  These menu options are used in connection with ASCET-RP. They are described in the ASCET-RP user's guide.

– *Edit Component*

  Opens the specification editor for an included component

– *Replace Component*

  Replaces a component with another component. The name of the old component remains.

– *Show Path*

  Displays the path of an included component.

– *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard
  (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path
  Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

– *Notes*

  Opens the notes editor for an included component.

– *Edit Distribution*

  Opens an editor for distribution (only for group tables).

– *Edit Max Size*

  Opens an editor where you can specify the maximum size of an array, a matrix or a table.

- *Copy Elements (<*CTRL*> + <*C*>)*

  Copies an element from the "Elements" list.
- *Paste Elements (<*CTRL*> + <*V*>)*

  Pastes a copied element into the "Elements" list.
- *File Out Data*

  Writes the data from an array or a table to a file.
- *File In Data*

  Reads the data for an array or a table from a file.
- *Export Data*

  Exports the data set of an included component.

• **Edit**

- *Element / Data / Implementation*

  Corresponds to the **Edit** / **Edit Data** / **Edit Implementation** entry from the context menu of the element list.
- *Cut (<*CTRL*> + <*X*>)*

  Cuts the highlighted text.
- *Copy (<*CTRL*> + <*C*>)*

  Copies the text highlighted.
- *Paste (<*CTRL*> + <*V*>)*

  Inserts code from the Clipboard
- *Find / Replace (<*CTRL*> + <*F*>)*

  Opens the editor for finding and replacing C code
- *Select All (<*CTRL*> + <*A*>)*

  Highlights all the code of the current method or current process
- *Load from file*

  Import/insert from a file.
- *Store to file*

  Code is written to an external file.
- *Print*

  Prints out the current code
- *Printer Setup*

  Opens the "Printer Selection" window for printer selection and setup.

  - *Save* (<CTRL> + <S>)

    Saves a method or process

- **View**
  - *Show/Hide*

    Shows/hides several parts of the editor or the component. (*Element List* → "Elements" pane, *Diagram List* → "Diagrams" pane).

- **Code Variants**
  - *Copy C-Code to*

    The code is copied (without any changes) to the selected target/ arithmetic combination.

  - *Copy C-Code from*

    Copies a Target/Arithmetic combination.

4.3.1 Specifying Components in C Code

One method called `calc` is created by default. Creating methods and processes, as well as specifying their interfaces, is the same as in the block diagram editor and is described in "Defining a Component Interface" on page 192. Creating of basic elements (cf. page 207) and enumerations (cf. page 208), including components as complex elements (cf. page 217) and editing notes (cf. page 220) also works the same way as in block diagrams.

Specifying a method or process:

- In the "Diagrams" pane, select the method or process you want to specify.

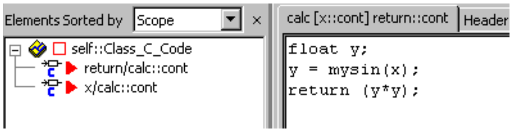
  The name of the method appears on the left tab.

- From the "Target" combo box, select a target.

  All targets installed on your ASCET system are available in the list.

- From the "Arithmetic" combo box, select an arithmetic for the code.
- Click on the "Header" tab.
- Enter the header information for the class or method, e.g. local variables, include statements or macros.
- Click on the "*<method name>*" tab.

- Enter the code for the method.



```
float y;
y = mysin(x);
return (y*y);
```

Basic elements defined in the "Elements" pane can be referred to simply by typing their names. This is however only the case if no referenced components are used within the component.

- Repeat for all methods or processes.

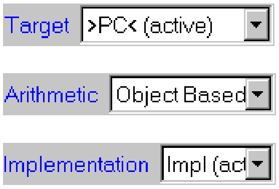
As C code is always platform-dependent, you have to specify the code of a C code component for each platform and each piece of arithmetic individually. However it is easy to copy code developed for one target/arithmetic combination for use with another one.

To copy C code for single classes or modules:

To copy existing C code of a single class or module to another target, experiment type, or implementation, use one of the two possibilities described here.

1. Use the menu item **Code Variants** → **Copy C-Code To**.

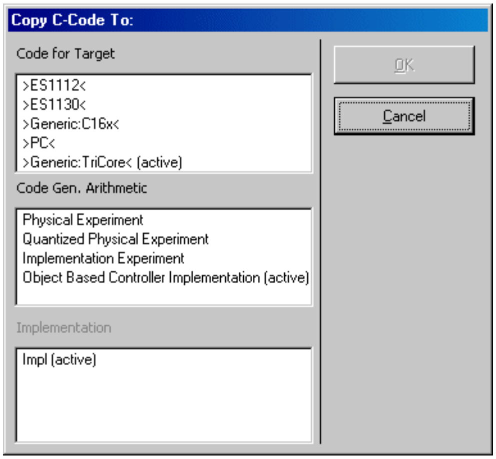
- Open the module/class in the C code editor.



Target: >PC< (active)
Arithmetic: Object Based
Implementation: Impl (act

- In the "Target" combo box, select the target the C code was written for.
- In the "Arithmetic" combo box, select the experiment type the C code was written for.
- In the "Implementation" combo box, select the implementation the C code was written for.
- Now select **Code Variants** → **Copy C-Code To**.

- The "Copy C Code to" dialog box opens.


Copy C-Code To:

Code for Target

>ES1112<
>ES1130<
>Generic:C16x<
>PC<
>Generic:TriCore< (active)

Code Gen. Arithmetic

Physical Experiment
Quantized Physical Experiment
Implementation Experiment
Object Based Controller Implementation (active)

Implementation

Impl (active)

- Select a target in the "Code for Target" pane.
- In the "Code Gen. Arithmetic" field, select the appropriate experiment type.

  All experiments can evaluate an implementation. All implementations defined are shown in the "Implementation" pane.
- Select an implementation.

  Once you have completed the selection, the **OK** button is activated.
- Click on **OK**.

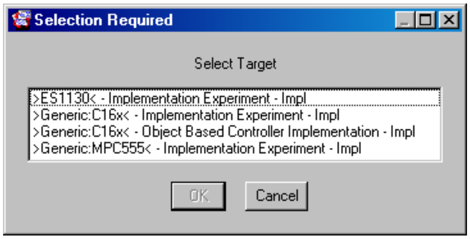
  The C code is copied to the selected controller.

2. Use the menu item **Code Variants** → **Copy C-Code From**.

   - In the C code editor, use the "Target", "Arithmetic", and "Implementation" combo boxes to set up the target you are using with the appropriate experiment type and implementation.

- Now select **Code Variants** → **Copy C-Code From**.

  The "Selection Required" window opens.



>ES1130< - Implementation Experiment - Impl
>Generic:C16x< - Implementation Experiment - Impl
>Generic:C16x< - Object Based Controller Implementation - Impl
>Generic:MPC555< - Implementation Experiment - Impl

- Choose the target, experiment type, and implementation you want to copy the code from, and click **OK**.

  The C code is copied to the current target.

To edit the code for a method or process:

- Highlight the section of code you want to edit

or

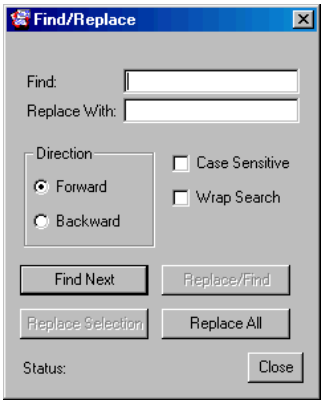
- select **Edit** → **Select All** to highlight the entire code of the current method or process.
- Select **Edit** → **Cut** to remove the code highlighted and store it on the clipboard.
- Select **Edit** → **Copy** to copy the highlighted code to the clipboard.
- Select **Edit** → **Paste** to paste the code from the clipboard.

The text you cut or copy is stored on the Windows clipboard and can be exchanged between other applications (e.g. C development systems). You can also copy and paste code between methods or processes, or between components.

To search and replace C Code:

- Select **Edit** → **Find/Replace**.

  The "Find/Replace" dialog box opens



- Type the text you want to find into the "Find" field.

  You can use wild cards: # for character, * for a string of characters.

- Type the text that you want to replace the search text with into the "Replace With" field.
- Click the **Find Next** button to find the next occurrence of the search text.

  The search text is found and highlighted. If the search text is not found, the message `String Not Found` is displayed in the status line at the bottom of the dialog box.

- Click **Replace Selection** to replace the highlighted text with the text in the "Replace With" field.

  This button is only available if the text has been found.

- Click **Replace/Find** to replace the highlighted text and find the next occurrence.
- Click **Replace All** to replace all occurrences.

- Click **Close** to close the "Find/Replace" dialog box.

Customizing the search criteria:

- Click the **Forward** option to search forward from the insertion point.

or

- Click the **Backward** option to search backward from the insertion point.
- Tick the **Case Sensitive** box to make the search case sensitive.
- Tick the **Wrap Search** box to make the search wrap around the text.

  If this option is checked, the search starts from the insertion point in the direction specified, and continues from the beginning or end of the text, back to the insertion point.

Saving a method or process:

- Select **Edit** → **Save**

or

- press <CTRL> + <S> to save the current method or process.

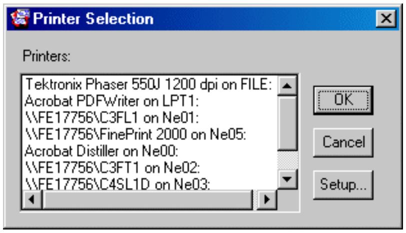
  This command is only available if the current method or process has been changed since it was last saved. If you switch between methods without saving, you are asked whether you want to save the current method.

Printing the C code:

- Select **Edit** → **Print** to print the code to your default printer.

- Select **Edit** → **Printer Setup** to select a printer.

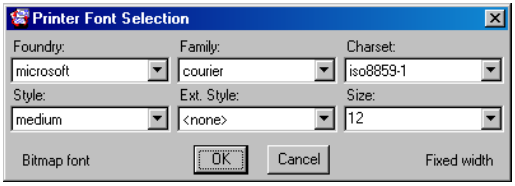
  The "Printer Selection" window opens.



Tektronix Phaser 550J 1200 dpi on FILE:
Acrobat PDFWriter on LPT1:
\\FE17756\C3FL1 on Ne01:
\\FE17756\FinePrint 2000 on Ne05:
Acrobat Distiller on Ne00:
\\FE17756\C3FT1 on Ne02:
\\FE17756\C4SL1D on Ne03:

- Select a printer in the "Printers" field.
- Click on **Setup** to change the printer setup.
- In the printer setup window, make the settings.
- Click **OK** to accept the selection.

  The "Printer Font Selection" window opens.



Foundry: microsoft
Family: courier
Charset: iso8859-1
Style: medium
Ext. Style: <none>
Size: 12
Bitmap font
Fixed width

- Select the font you want to use.
- Click **OK** to accept the selection.

4.3.2 External Editor

The editor available for creating C code is limited and can only carry out simple operations. Another way of creating code is to select a user defined external editor (e.g. Notepad, Codewright, etc.). As the external editor is connected via a file system, files with the suffixes "`*.c`" and "`*.h`" must be associated with the editor in the Windows configuration. Without this association, the external editor cannot be opened from the ASCET environment. However, an error message indicates that the association is missing.

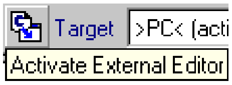
After using the button for the external editor, the view of the C code editor is changed. It is divided into an upper section listing all the available methods or processes of a component and a lower section showing the C code.

When the external editor is opened, the code for a method or a process is written to one or more files which are stored in a temporary directory. After you have finished working, the files first have to be saved in the external editor, before they can be returned to the ASCET environment. Closing the external editor from the ASCET environment writes the stored code changes from the external editor to the ASCET environment, at the same time reading and deleting the temporary file. Code changes made afterwards in the external editor environment can no longer be returned to the ASCET environment. If you need to re-edit the code, you must call the external editor from the ASCET environment again.

Note

In order to return code changes made in the external editor to the ASCET environment, you must save the code in the external editor first.

Opening an external editor:

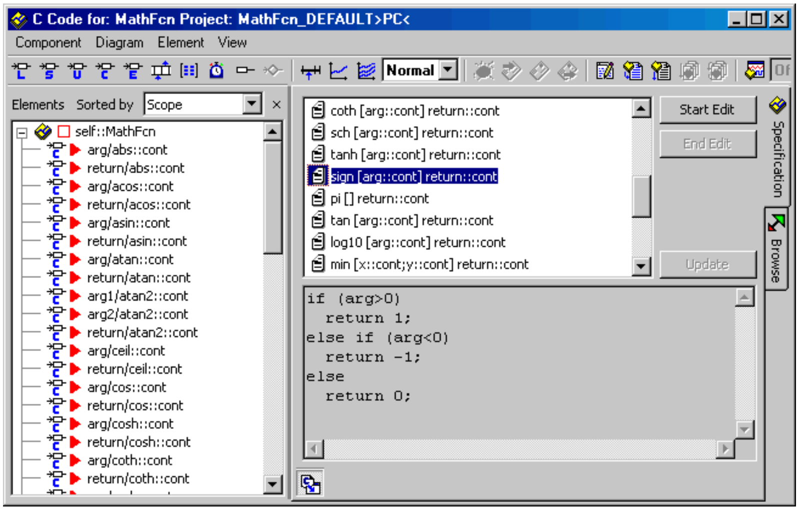


- Click on the **Activate External Editor** button at the bottom of the editor window.

  The view of the C code editor changes. The "Diagrams" pane disappears, the code pane is replaced by a selection field and a text field. All defined methods or processes are listed in the selection field.

- From the selection field, select the method or process for which you wish to change the code.

  The valid code for the selected method or process selected is displayed in the text field for checking purposes.


C Code for: MathFcn Project: MathFcn_DEFAULT>PC<
Component Diagram Element View
Normal
Elements Sorted by Scope
Start Edit
End Edit
Update
Specification
Browse

self::MathFcn
arg/abs::cont
return/abs::cont
arg/acos::cont
return/acos::cont
arg/asin::cont
return/asin::cont
arg/atan::cont
return/atan::cont
arg1/atan2::cont
arg2/atan2::cont
return/atan2::cont
arg/ceil::cont
return/ceil::cont
arg/cos::cont
return/cos::cont
arg/cosh::cont
return/cosh::cont
arg/coth::cont
return/coth::cont

coth [arg::cont] return::cont
sch [arg::cont] return::cont
tanh [arg::cont] return::cont
sign [arg::cont] return::cont
pi [] return::cont
tan [arg::cont] return::cont
log10 [arg::cont] return::cont
min [x::cont;y::cont] return::cont

```
if (arg>0)
  return 1;
else if (arg<0)
  return -1;
else
  return 0;
```

- Click on **Start Edit** to open the external editor with the code of the highlighted method or process.

sign [arg::cont] return::cont

  After the external editor has been opened, the method or process in the upper dialog field is marked with a red dot in the symbol.

- Edit the code in the external editor.

Closing an external editor:

- Save the newly created or modified code in the external editor.
- In the C code editor, click on the **Update** button.

  The current version of the code modifications in the external editor is transmitted to the ASCET environment. The code changes are displayed in the text field.

This does not close the connection to the external editor. Thus, you can continue working in the external editor, making changes and, when you have saved these changes, updating them in the ASCET environment.

- Click on the **End Edit** button to stop working with the external editor.

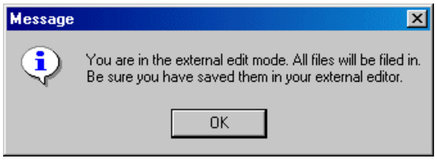
  The connection to the external editor is closed, the red dot in the method or process symbol is deleted. Any changes made afterwards in the open external editor are *not* applied.

- Close the external editor.

To end the external editor mode:

- Click on the **Activate External Editor** button.

  When you did not click on **End Edit** first, a warning is displayed. You are informed that the content of the external editor is read to the ASCET environment.



You are in the external edit mode. All files will be filed in. Be sure you have saved them in your external editor.

  – If necessary, save the file in the external editor to include the most recent alterations.

  – Confirm the warning.

  The external editor mode is ended, and the original view of the C code editor is restored.

4.3.3 Using the External Source Editor

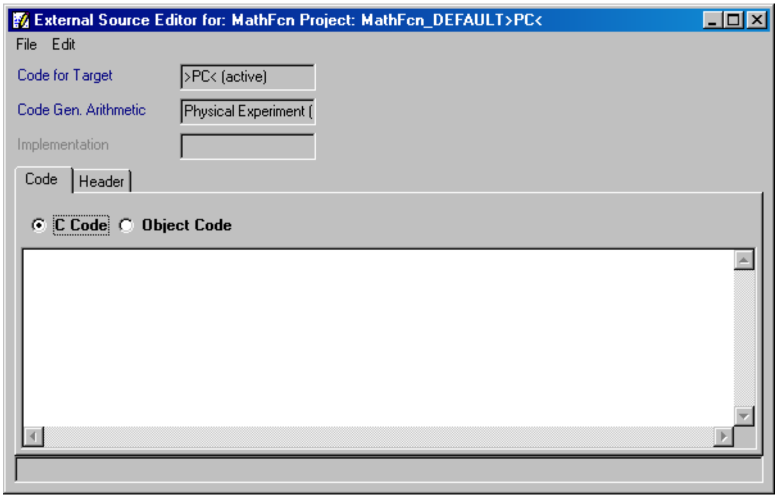
The external source editor allows you to integrate existing C-programs and use them as part of the specification of a C code component. You can include either C-source or object code. All the functions contained in an external source file (for both source and object modules) have to be declared in a header file that is also part of the component.

External modules also have to be specified separately for each target and arithmetic. Target and arithmetic selection is as described earlier in this chapter.

Opening the external source editor:

- Click the **External Source Editor** button.

  The external source editor opens



External Source Editor for: MathFcn Project: MathFcn_DEFAULT>PC<

Code for Target: >PC< (active)

Code Gen. Arithmetic: Physical Experiment (

Implementation:

Integrating an external source file:

- Click the "Code" tab.

  You can only have one external source file in a component; this can be either a C code module or an object code module.
- Click on the **C Code** or **Object Code** option.
- Enter the functions of the external module (option **C Code** only)

or

- select **Edit** → **Load from File**.

  The "File Selection" dialog box appears.
- Select the external C code or object module.

- Click on **Open**.

  The contents of the selected file is written in the text pane; the original file is not affected.
- Select **Edit** → **Store to File**.

  The "File Selection" window opens. The content of the text pane is written to the selected file.
- Select **Edit** → **Save** to save the external module.

The same editing commands are available in the external source editor as in the C code editor.

If an external class contains an external source module, there also has to be a header file that declares the functions in the source module.

Adding a header file to the source module:

- Click on the "Header" tab.

  You can add the header file before or after the source module.
- Type in the text of the header file or load it as described above.
- Tick the **Use header global** tick box to use the same header file for all your C code.

Header files thus marked are included in each generated `*.c` file.

Testing the external module:

- Select **File** → **Compile C-Code**.

  This command calls up the currently selected compiler and compiles the `.c` and `.h` files in the external module. This does not result in any executable code, but merely serves as a test whether the source code is syntactically correct. If it is not, the compiler will display the appropriate error messages.

4.4 The ESDL Editor

The ESDL editor is used to specify components in ESDL code. ESDL components can either be classes, modules or continuous time blocks. A class in ESDL code works in the same way as a class specified as a block diagram. When it is used by another component, it looks and behaves just like a class specified as a block diagram.

This chapter explains how to create classes and modules in ESDL code. Before reading this chapter, you should be familiar with creating classes as block diagrams, as explained in chapter "The Block Diagram Editor" on page 178. The ESDL language itself is described in chapter "Body Specification in ESDL" in the ASCET reference guide.

Creating a class or module in ESDL code:

- In the Component Manager, select a folder for the new component.
- Select **Insert** → **Class** → **ESDL** or **Module** → **ESDL**.

Or

- use the **Insert Class - <Type>** or **Insert Module - <Type>** buttons.
  - Use one of the arrow buttons to select the type **ESDL**.
  - Click on **Insert Class - ESDL** or **Insert Module - ESDL**.

  A new component is created in the selected folder.
- Type in a name for the component and press <Enter>.
- On the menu bar, select **Component** → **Edit Item**

or

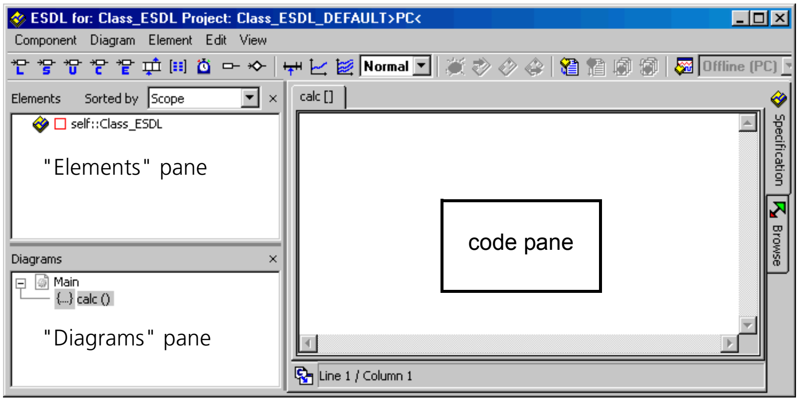
- press <ENTER>

or

- in the "1 Database" list, double-click on the component name.

  The ESDL editor for the new component opens.

As in the block diagram editor, components specified in ESDL code have methods or processes, which in turn can contain elements. The same kinds of elements are available, the interface and the methods are defined in the same way. The diagrams feature is also available, but it only serves to structure the methods of the components into groups of public and private methods. The ESDL code of the class is displayed one method/process at a time in the code pane of the ESDL editor.



The facilities in the menu bar correspond to the features in the C code editor (see page 310), with two exceptions: the menu options **Diagram** → **Analyze Diagram** and **View** → **Show/Hide** → **Impl. Casts in Element List** are available in the ESDL editor.

4.4.1 Specifying Classes in ESDL Code

Creating ESDL code:

- Add the methods needed, as described in "Defining a Component Interface" on page 192.
- Select a method in the "Diagrams" pane.
- Create the elements you want by clicking on the element buttons and typing in the names for the elements.
- Type the code for the method into the code pane.

  You can use an external editor the same way as in the C code editor (cf. chapter 4.3.2).

- Select **Edit** → **Save**

or

- press <CTRL> + <S> to save the code of the current method.

  You are asked whether you want to save every time you switch to another method.

In a textual specification there are no sequence calls; the order in which statements are evaluated is determined by their order in the source code. Elements and other methods are referred to in the code by their names.

To use implementation casts in ESDL:

Unlike the scenario in the block diagram editor, implementation casts can only be added to the ESDL editor using a button.

- Add the required number of implementation casts using the **Implementation cast** button.
- Use the implementation casts in ESDL code in accordance with the rules listed in the "Implementation Casts in ESDL" chapter of the ASCET reference guide.
  - Implementation casts are referenced by their names.
  - They are always enclosed in parentheses.
  - They are immediately in front of the element they refer to.
  - If an implementation cast refers to the result of an operation, this operation has to be enclosed in parentheses. The operation can be part of a larger calculation.

Note

When you apply an implementation cast to a logical variable or expression, an error message is generated during code generation.

As in the block diagram editor, you can show and hide the display of implementation casts in the "Elements" list using **View** → **Show/Hide** → **Impl. Casts in Elements List** (see page 249).

Editing ESDL code:

It is possible to cut, copy and paste ESDL code between methods or within a method.

- Select the code you want to edit.

Or

- Select **Edit** → **Select All** to select all the code of the current method.
- Select **Edit** → **Cut** to cut the highlighted text or **Edit** → **Copy** to copy it to the clipboard.
- Click where you want to add the text.
- Select **Edit** → **Paste** to paste the text from the clipboard.

Searching/replacing and printing ESDL code:

A sophisticated search and replace feature is available in the ESDL editor. It works in the same way as the one for the C code editor and is described in detail in section "To search and replace C Code:" on page 320.

You can print out ESDL code the same way as C Code (cf. "Printing the C code:" on page 321).

4.4.2 Specifying Modules in ESDL

Specifying modules in ESDL code works in the same way as specifying classes, except that processes are defined rather than methods. Furthermore, it is possible to define messages in modules. When specifying classes, the **message** buttons are greyed out in the ESDL editor, when specifying modules they are active.

4.4.3 Analyzing ESDL Components

After you have created a block diagram, you will usually want to experiment with it to see whether it works as intended. The procedure is the same as for block diagrams; see "Analyzing Components" on page 258.

Note

For an individual ASCET module, code can be generated and simulated without project context only in the physical experiment. For the other code generators the module must be integrated into a project. A so-called default project can be defined for each class or module for that purpose. This is the only way to access the implementation information. Without project context, the conversion formulas as well as all implementations of imported entities are missing.

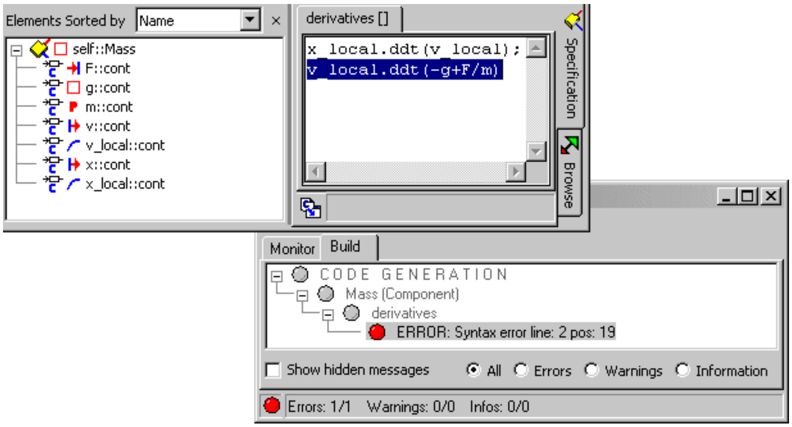
As for block diagrams, you do not have to generate the complete code each time, there is an analysis function.

To analyze a diagram:

- Select **Diagram** → **Analyse Diagram** to analyze the current ESDL component.

  The diagram is checked for syntactic errors. Detected errors are listed in the ASCET monitor window.

- Click on an error message to have the error highlighted automatically in the ESDL editor.



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

CODE GENERATION
Mass (Component)
derivatives
ERROR: Syntax error line: 2 pos: 19

Errors: 1/1 Warnings: 0/0 Infos: 0/0

4.5 Specifying Continuous Time Blocks

Continuous time blocks (CT blocks) are used to describe models of the technical processes controlled by ASCET embedded software specifications. Entire control loops can be modelled and tested within ASCET in this way. CT blocks are discussed in detail in chapter "Continuous Time Systems" in the ASCET reference guide.

CT blocks are specified in the same way as other components, i.e. they can be specified either as block diagrams, C code or ESDL code. There is, however, a difference in functionality between CT blocks specified as block diagrams and those specified in code.

Basic continuous time blocks specified in code are used to model basic physical components, such as wheels, brakes, or hydraulic conduits. These components are typically described in terms of differential equations, which are more easily specified in code than as block diagrams. They are then combined into larger assemblies with continuous time structure blocks specified as block diagrams.

It is possible to reference standard ASCET classes, but this only makes sense if those classes are used as records, i.e complex variables. It is not possible to use algorithms specified as standard classes in continuous time blocks.

This chapter shows how to specify continuous time blocks both as block diagrams and in code. The editors used are the same ones as for standard -ASCET classes, with slight variations. Only the differences from standard class specification are discussed here, so you should be familiar with the chapters "The Block Diagram Editor" on page 178, "The C Code Editor" on page 308 and "The ESDL Editor" on page 328.

4.5.1 Continuous Time Blocks as Block Diagrams

To specify a continuous time block:

- In the Component Manager, select a folder for the new block.
- Select **Insert** → **Continuous Time Block** → **Blockdiagram**.
- Type in a name for the component and press <ENTER>.
- From the menu bar, select ComponentEdit Item

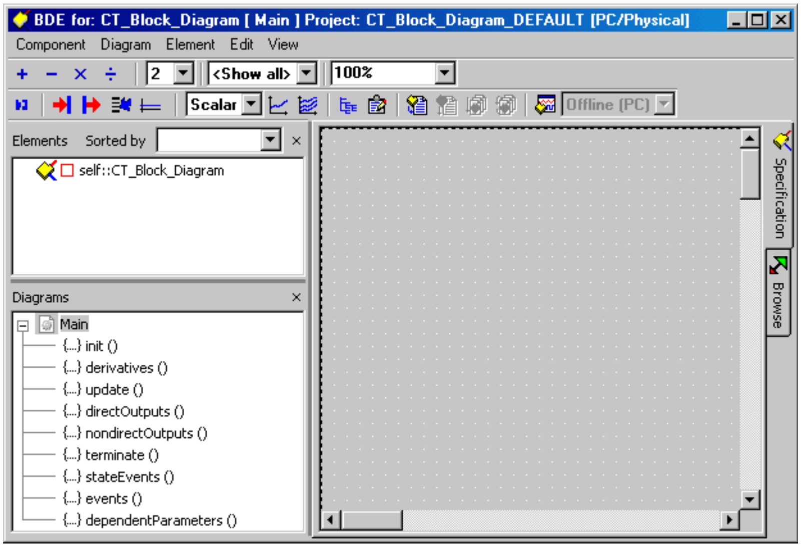
or

- press <ENTER>

or

- in the "1 Database" list, double-click on the component name.

  The block diagram editor for CT blocks opens.



As mentioned elsewhere, a CT block specified as a block diagram consists mainly of basic blocks which are then connected graphically to form a larger assembly. Specifying the diagram is very similar to specifying classes, the most important differences are listed below:

- There are only four operators available: addition, subtraction, multiplication and division. If more complex non-linear operators are required, they can be specified as separate blocks.
- The basic elements that can be defined in a CT block are different, with the exception of characteristic lines and fields. These work in the same way as in classes. The basic elements that can be defined in CT blocks are discussed in section "Block Interfaces" in the ASCET reference guide.
- There is only one diagram in a CT block; additional diagrams cannot be defined. The diagram can be hierarchical, however.
- Interfaces need not be defined. A set of methods is predefined when the CT block is created. These cannot be changed and additional methods cannot be defined.

- Only classes and other CT blocks can be referenced. Classes can only be used to define records, not to specify functionality. When a class is referenced, you are asked whether it is to be used as an input or an output.
- There are no sequence calls in a CT block. The order for evaluation is determined automatically.

Apart from these considerations, all the block diagram editor features work in the same way as for classes, and all the commands available here have the same effect.

4.5.2 Continuous Time Blocks as C Code

The considerations applying CT blocks as block diagrams are valid for CT blocks as C code. too, i.e. everything works in the same way as for classes, except for the points listed above. One difference between continuous time blocks specified as block diagrams and as C code is that the basic elements that can be defined are different. CT blocks in C code are discussed in section "Modeling in C" in the ASCET reference guide.

CT blocks specified as C code are used mainly to specify basic components for a process model. The differential equations used to model these components can be specified more quickly and concisely in code than in a block diagram.

To create a continuous time block:

- In the Component Manager, select the folder for the new block.
- Select **Insert** → **Continuous Time Block** → **C Code**.
- Type in a name for the component and press <ENTER>.
- Select **Component** → **Edit Item**

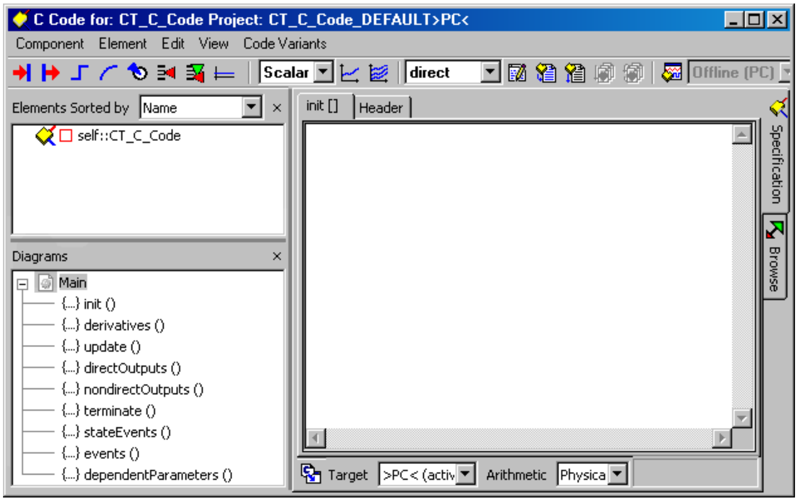
or

- press <ENTER>

or

- in the "1 Database" list, double-click on the component name.

  The C code editor for CT blocks opens.



- Select `direct` or `nondirect` from the right combo box in the button bar.
- Type in the code or import external modules as required.

  Using the C code editor is described in detail in chapter 4.3.

CT blocks specified in C code support either direct or nondirect outputs, but not both. You can set this in the "Block Behavior" combo box. If you selected `direct`, only the `directOutputs[]` method is available, with `nondirect` only the `nondirectOutputs[]` method is available. See section "Algebraic Loops" in the ASCET reference guide for details on direct and nondirect outputs.

4.5.3 Continuous Time Blocks in ESDL

Again, specifying CT blocks in ESDL is similar to specifying classes with the exception of the differences listed above. The same basic elements are available in ESDL as for CT blocks specified in C code; the two types of blocks serve the same purpose.

To create a continuous time block:

- In the Component Manager, select the folder for the new block.

- Select **Insert** → **Continuous Time Block** → **ESDL**.
- Type in a name for the component and press <ENTER>.
- Select **Component** → **Edit Item**

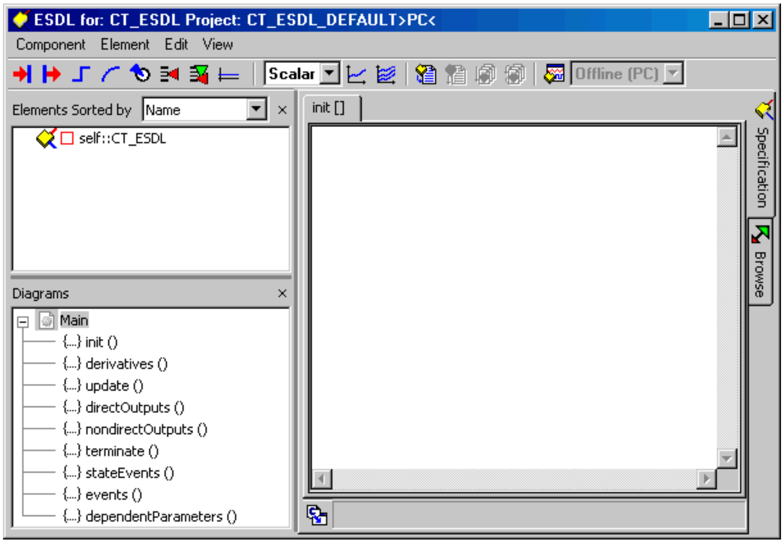
or

- press <ENTER>

or

- in the "1 Database" list, double-click on the component name.

  The ESDL editor for CT blocks opens.



- Type in the code for the block.

Using the ESDL editor is described in detail in "The ESDL Editor" on page 328.

4.5.4 Experimenting with Continuous Time Blocks

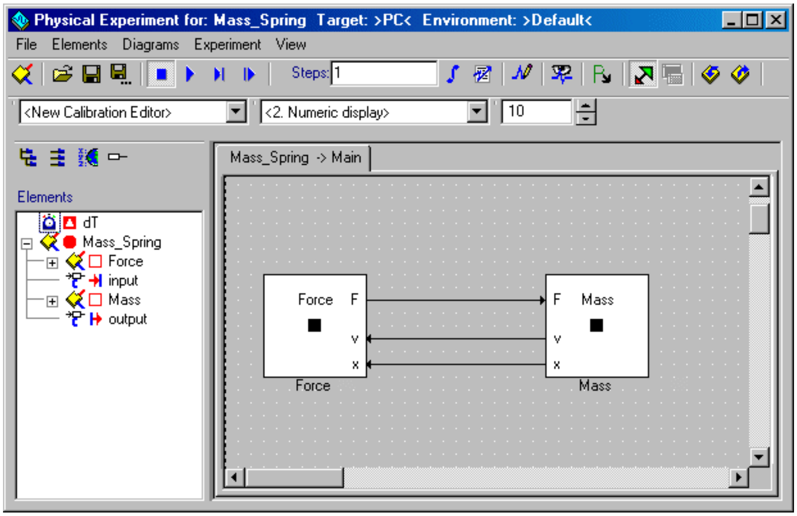
It is possible to run offline experiments with individual CT blocks. The blocks can be complex, i.e. they can reference other blocks. It is not possible, however, to conduct hybrid experiments in this way. Hybrid experiments consist of both classes and CT blocks and are used to simulate control loops consisting of a controller and a process model. These loops can be experimented within projects.

Experimenting offline with individual CT blocks works in the same way as experimenting with classes (see "Analyzing Components"), except that there is no event generator. A solver has to be specified instead.

To experiment with a continuous time block:

- In the Component Manager, select the block you want to experiment with.
- Select **Build** → **Experiment** → **Offline** to open the experimentation environment for the block.

  You can also start the experiment from the editor.



- Set up the data generator, measurement and calibration windows and run the experiment.

Except for the solver configuration, the offline experimentation environment works as normal. Using the experimentation environment is described in "The Experimentation Environment" on page 561.

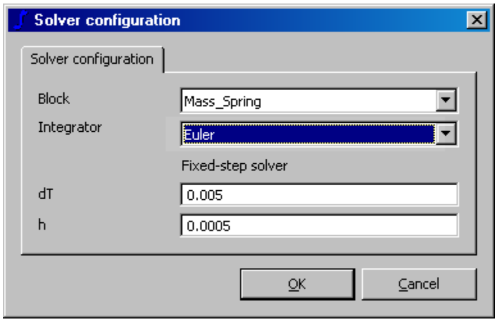
To configure the solver:

- Select **Experiment** → **Open CT Solver**.

or

- Click on the **Open CT Solver** button.

  The "Solver Configuration" dialog box opens.



Block: Mass_Spring
Integrator: Euler
Fixed-step solver
dT: 0.005
h: 0.0005

- Select the solver from the "Integrator" combo box.

  Depending on the selected method, different parameter fields are shown.

- Set the dT and h parameters.

Note

With online experimentation, (cf. "Online and Offline Experimentation" on page 437), dT *is set in the "OS" tab of the project editor. The input field in the "Solver Configuration" window is disabled; changes during measurement are not possible.*

- Click **OK**.

The solver is always specified for the entire model (global integration). When experimenting in a project context, a separate solver can be specified for each block separately (multirate). It is possible to change the solver during the experiment. The following solvers are available:

- Adams-Moulton 2
- Euler
- Mulstep 2

- Heun
- Runge-Kutta 4
- Variable-step Dormand/Prince RK5
- Variable-step Calvo 6(5)
- Variable-step Dormand/Prince RK8
- Variable-step implicit RK2
- Variable-step implicit RK4
- Variable-step implicit Gear 1
- Variable-step implicit Gear 2

For the first five solvers, only the *dT* and *h* parameters can be set. There are additional parameters for the Gear 4 solver. The solvers are described in "Solving Differential Equations – Integration Algorithms" in the ASCET reference guide. The parameters have the following meaning:

- *dT* is the communication time frame, i.e. the interval in which the block communicates with the outside.
- *h* is the integration step size, i.e. the interval for internal calculations.
- *Initial h* is the integration step size for solvers with variable step size.
- *Minimum h* and *Maximum h* are upper and lower limits for the h parameter which is calculated according to the model dynamics when the variable step size solvers are used. *Maximum h* must be less than or equal to *dT*.
- *Relative error* and *Absolute error* are the relative error and abslute error allowed for the calculation of h; available for solvers with variable step size.
- *Max. iterations* is the maximum number of iterations used to calculate the h parameter. Once the specified number of steps has been carried out, the h value at that point will be used.

Monitoring the Cycle Time

In the "OS" tab of the project editor, ASCET offers the possibility to measure the cycle time of a timer task (see "The Monitoring Option" on page 400).

To activate the cycle time monitoring:

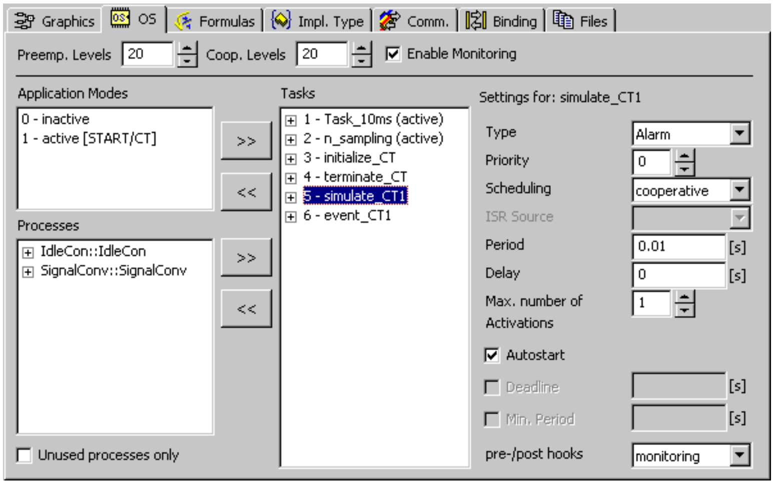
- Open a CT block.

- Select **Component** → **Default Project** → **Edit** to open the default project for the CT block.

Or

- Open a hybrid project (cf. chapter 4.8.7 "Hybrid Projects" and "Projects and Hybrid Projects" in the ASCET reference guide).
- Activate the "OS" tab of the project editor.

  In the "Tasks" list, you find the tasks automatically defined for the CT block (cf. section "Combining Continuous Time Blocks With Modules" in the ASCET reference guide).



Preemp. Levels 20   Coop. Levels 20   [✓] Enable Monitoring

Application Modes
0 - inactive
1 - active [START/CT]

Processes
IdleCon::IdleCon
SignalConv::SignalConv

[ ] Unused processes only

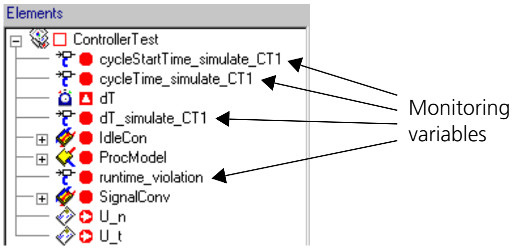
Tasks
1 - Task_10ms (active)
2 - n_sampling (active)
3 - initialize_CT
4 - terminate_CT
5 - simulate_CT1
6 - event_CT1

Settings for: simulate_CT1

Type	Alarm
Priority	0
Scheduling	cooperative
ISR Source	
Period	0.01 [s]
Delay	0 [s]
Max. number of Activations	1
[✓] Autostart	
[ ] Deadline	[s]
[ ] Min. Period	[s]
pre-/post hooks	monitoring

- Select the `simulate_CT1` task.
- From the "Pre-/post hooks" combo box, select `Monitoring`.

  The monitoring variables for each task are generated during the next code generation for an experiment. You can show them in the measure windows or write them to the data logger (offline experiment only, cf. "The Data Logger" on page 597).



For CT blocks, you have to keep the following in mind. The simulation of projects containing CT blocks allows the setting of two parameters for a selected solver:

- the external communication interval `dT` (cf. "External Communication Interval dT" in the ASCET reference guide)
- and the integration step size `h` (cf. "Integration Step Size h" in the ASCET reference guide).

The cycle time for the simulation task `simulate_CTn` sums up all integration steps performed during one `dT` step. Thus, the higher the ratio `dT/h` is, the higher is the cycle time. A correction factor cannot be specified, however, because other factors as the size and type of the model contribute as well.

A CT block specified with Simulink, on the other hand, does not distinguish between `dT` and `h`; both have the same value. If such a CT block is imported into ASCET, you have no possibility to change the solver or `h` within the experiment. To obtain comparable cycle times for CT blocks specified with ASCET or Simulink, the integration step size of the Simulink model has to be chosen for both `dT` and `h` in the ASCET block.

4.6 The Boolean Table Editor

A *Boolean table* mirrors logical behavior. Its columns contain several logic inputs and outputs whose values can be specified. The lines, so-called *combinations*, contain the various combinations of input values and the respective output values. In other components, a Boolean table is used as logic connection. You can attach notes to a Boolean table, like other components (cf. page 220).

With the Boolean table editor, you can specify the Boolean table. ESDL code describing the specified output behavior is generated automatically when the editor is closed. The inputs are generates as logical variables; for each output, a respective method is created.

Creating a Boolean table:

- In the Component Manager, select the folder for the new table.
- Select **Insert** → **Boolean Table**

or

- click on the **Insert Boolean Table** button.

  A new Boolean table is created.
- Enter a name for the component and press <ENTER>.
- From the menu bar, select **Component** → **Edit Item**

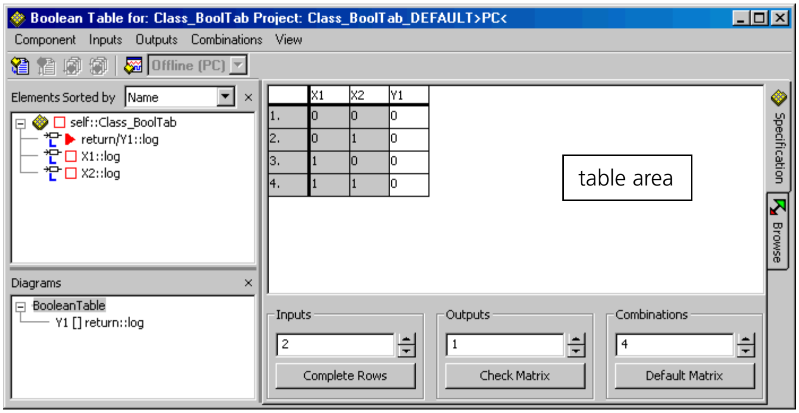
or

- press <ENTER>

or

- Double-click on the selected element.

  The Boolean table editor is opened.



Elements Sorted by Name

self::Class_BoolTab
return/Y1::log
X1::log
X2::log

BooleanTable
Y1 [] return::log

	X1	X2	Y1
1.	0	0	0
2.	0	1	0
3.	1	0	0
4.	1	1	0

Inputs: 2
Outputs: 1
Combinations: 4

Manipulations of the interfaces are not possible in this editor. In addition, inputs and outputs cannot be edited from the "Elements" pane. Therefore, most menu items from the other editors do not exist here; the browse display mode has also been omitted.

The background colors of the table can be selected in the ASCET option window (see "Options for Table Editors" on page 57).

Description of the Menu Options

- **Component**
  - *Clean code generation directory*

    Deletes all files in the code generation directory.
  - *Touch*

    Forced regeneration during the next code generation
    (*Flat* → the edited component,
    Recursive → disabled for Boolean tables).
  - *Generate Code*

    Generates the code for a component.
  - *Compile*

    Compiles the generated code.
  - *Open Experiment*

    Starts an experiment.
  - *Default Project*

    Allows editing the default project used for experimenting with components
    (*Edit* → opens an editor for the default project,
    Resolve Globals → automatically creates a global element for each imported element in the component for which there is no exported element,
    Delete Unused Globals → deletes unused global elements).
  - *Edit Layout*

    The layout editor opens.
  - *Edit Data*

    Opens the data editor for a component. Search of component data is possible.

– *Edit Implementation*

  Opens the implementation editor. Search of component implementations is possible.

– *Edit Notes*

  Opens the Notes editor - you can make notes about the component here.

– *Check Dependency*

  Checks whether the allocation of formal parameters to the model parameters is correct.

– *Export Data*

  Exports a component data set.

– *Show Path*

  Shows the path of an included component.

– *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard
  (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path
  Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

– *File out Generated Code*

  Saves the code generated in the file system
  (*Flat* → the edited components,
  Recursive → the referenced components also)

– *View Generated Code*

  Generates the code for the component and displays it in a text editor.
  Generates the code for the component and displays it in a text editor.
  The text editor can be selected in the ASCET options window, "ASCII Editor" node (cf. page 60).

– *File Out Generic Code For External Make*

  Saves the code generated in any directory for processing at a later point using external tools, e.g. an external Make/Build process.

- *Exit*

  Exits the Boolean table editor.

- **Inputs**
  - *Add*

    Adds a new input and places it behind the last existing input.
  - *Insert*

    Adds a new input and places it to the left of the selected input.
  - *Delete*

    Deletes the selected input.
  - *Rename*

    Renames the selected input.
  - *Move Left*

    Moves the selected input to the left.
  - *Move Right*

    Moves the selected input to the right.
- **Outputs**

  The **Outputs** menu is used to edit outputs. The commands function in the same way as the **Inputs** menu.
- **Combinations**
  - *Add*

    Adds a new combination after the last line of the table.
  - *Insert*

    Adds a new combination and places it above the selected combination.
  - *Delete*

    Deletes the selected combination.
  - *Move Up*

    Moves the selected combination upwards within the table.
  - *Move Down*

    Moves the selected combination downwards within the table.

- **View**
  - *Show/Hide*

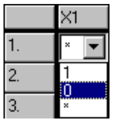
    Shows/hides several parts of the editor or the component. (*Element List* → "Elements" pane, *Diagram List* → "Diagrams" pane).

Specifying Boolean Tables

Two entries X1 and X2 and one output Y1 are created by default for a Boolean table, as well as four combinations (1 - 4) for the possible input settings. The output values of all combinations are set to 0 (false).

You can edit the input and output values of each combination.

Specifying inputs and outputs:



- In the table area, click on the input or output value you want to modify.

  A combo box appears.
- Select the value you want to assign from the combo box (0 - false, 1 - true).

Note

*Besides 0 and 1, entries can have the value * (undefined).*

If you want to add further inputs or outputs, proceed as follows:

Adding inputs or outputs:

- Select **Inputs** → **Add** or **Outputs** → **Add**

or

- right-click in the table area and select **Inputs** → **Add** or **Outputs** → **Add** from the context menu.

  One input or output is added.

Or

- Adjust the number of inputs or outputs in the "Inputs" or "Outputs" field with the **arrow** buttons.

  The specified number of inputs or outputs is created.
  Newly created inputs are set to * (undefined) by default, new outputs to `0` (`false`).

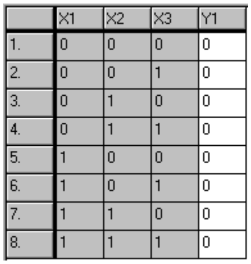
Combinations can be added by **Combinations** → **Add** or via the "Combinations" field.

You can now specify the input and output values as described above. Instead of editing each value by hand, you can generate a default matrix automatically.

Generate a default matrix:

- Create the desired number of inputs.
- Click on the **Default Matrix** button.

  Enough combinations are created to include every possible input setting. The number of outputs is unchanged; all output values are set to `0` (`false`). If, for example, you have created three inputs and one output, **Default Matrix** will give the following result:



	X1	X2	X3	Y1
1.	0	0	0	0
2.	0	0	1	0
3.	0	1	0	0
4.	0	1	1	0
5.	1	0	0	0
6.	1	0	1	0
7.	1	1	0	0
8.	1	1	1	0

Deleting inputs and outputs or combinations:

Inputs, outputs or combinations can be deleted as follows:

- Select tie input, output or combination you want to delete.
- In the table area, select the input, output or combination you want to delete.

- Select **Inputs** → **Delete**, **Outputs** → **Delete** or **Combinations** → **Delete**

or

- right-click in the table area and select **Inputs** → **Delete**, **Outputs** → **Delete** or **Combinations** → **Delete** from the context menu.

  The selected input, output or combination is deleted.

Or

- Adjust the number of inputs, outputs or combinations in the "Inputs", "Outputs" or "Combinations" field with the **arrow** buttons.

  The appropriate number of inputs, outputs or combinations is deleted.

Renaming inputs or outputs:

You can rename inputs and outputs. Combinations cannot be renamed.

- In the table area, highlight the input or output you want to rename.
- Select **Inputs** → **Rename** or **Outputs** → **Rename**

or

- Right-click on the highlighted input or output and select **Inputs** → **Add** or **Outputs** → **Add** from the context menu.

  A dialog window opens.

- Enter the new name and confirm with **OK**.

Shift Table Columns and Rows:

You can shift the columns containing the input and output values, as well as the combination rows.

- In the table area, highlight the input you want to shift.
- Select **Inputs** → **Move Left** or **Inputs** → **Move Right**

or

- right-click on the highlighted input and select **Inputs** → **Move Left** or **Inputs** → **Move Right** from the context menu.

  The *content* of the column is shifted to the left or right, respectively. The *names* of the columns keep their original order.

- Shift outputs with **Outputs** → **Move Left** or **Outputs** → **Move Right**.

Note

Inputs can only be exchanged with inputs, and outputs with outputs. It is not *possible to exchange the last input column with the first output column.*

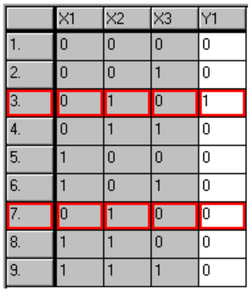
- Shift combinations with **Combinations** → **Move Up** or **Combinations** → **Move Down**.

If you have inserted the input and output values manually, or if you have deleted inputs or outputs, you should check the table for consistency.

Checking the Table:

- When you have set up the table according to your wishes, click on the **Check Matrix** button.

  If different output values are assigned to combinations with identical input values, the respective lines are marked with red borders.



	X1	X2	X3	Y1
1.	0	0	0	0
2.	0	0	1	0
3.	0	1	0	1
4.	0	1	1	0
5.	1	0	0	0
6.	1	0	1	0
7.	0	1	0	0
8.	1	1	0	0
9.	1	1	1	0

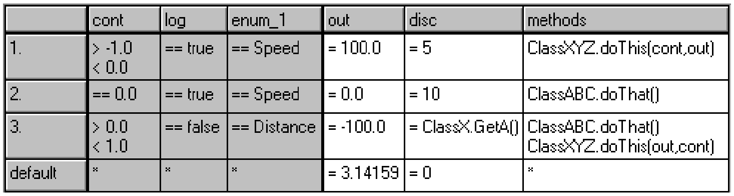
- Remove the redundant combinations with **Combinations** → **Delete**.

To experiment with a Boolean table, you can start the ecperiment via the **Open Experiment for selected Experiment Target** button, as with any other component.

4.7 The Editor for Conditional Tables

A *conditional table* maps logical behavior (in the same way as the Boolean table). Its columns and rows create a logical matrix with different conditions and instructions.

Basic elements can be used in a conditional table. It is also possible to access public methods of added classes. The functionality of a conditional table is the equivalent of an `If…Then…ElseIf…Then` construction. Every line corresponds to an `If` or `Elseif` query. Fig. 4-1 shows an example of a conditional table. The first three columns ("cont", "log" and "enum_1") contain the conditions linked by AND; the following columns ("out", "disc" and "methods") contain the relevant instructions.



	cont	log	enum_1	out	disc	methods
1.	> -1.0 < 0.0	== true	== Speed	= 100.0	= 5	ClassXYZ.doThis(cont,out)
2.	== 0.0	== true	== Speed	= 0.0	= 10	ClassABC.doThat()

3.	> 0.0
	< 1.0
		== false	== Distance	= -100.0	= ClassX.GetA()	ClassABC.doThat()
						ClassXYZ.doThis(out,cont)

default	*	*	*	= 3.14159	= 0	*

Fig. 4-1 Conditional Table – Example

The example in Fig. 4-1 corresponds to the following code:

```
if ((cont > -1.0) && (cont < 0.0) && (log == true)
                  && (enum_1 == Speed)){
   out = 100.0;
   disc = 5;
   ClassXYZ.doThis(cont,out);
} else
if ((cont == 0.0) && (log == true)
                  && (enum_1 == Speed)){
   out = 0.0;
   disc = 10;
   ClassABC.doThat();
} else
```

```
if ((cont > 0.0) && (cont < 1.0) && (log == false)
                 && (enum_1 == Distance)){
   out = -100.0;
   disc = ClassX.GetA();
   ClassABC.doThat();
   ClassXYZ.doThis(out,cont);
} else {
   out = 3.14159;
   disc = 0;
}
```

A conditional table is activated by calling a trigger method. On activation, the relevant row is detected and the instructions and calculations contained in it are executed.

Conditional tables are ESDL classes which are specified in a special *editor for conditional tables*. Conditions and instructions are entered in two separate areas of the table, condition and instruction area. Each area has a settable number of columns and its own freely selectable background color (see "Options for Table Editors" on page 57).

To create a conditional table:

- In the Component Manager, select the folder for the new table.
- Select **Insert** → **Conditional Table**

or

- click on the **Insert Conditional Table** button.
- Enter the component name and press <RETURN>.

  A new conditional table is created.
- Select **Component** → **Edit Item**

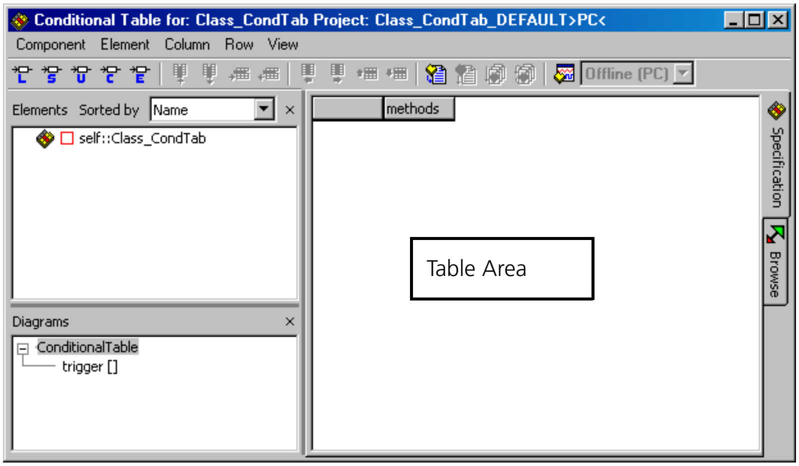
or

- press <RETURN>

or

- double-click the selected element.

  The conditional table editor opens.



The public trigger method `trigger` is created automatically in every case. It is mandatory and cannot be deleted or renamed. You cannot add additional methods or diagrams.

The "methods" column is also created automatically. It is always at the end of the instruction area and cannot be deleted or repositioned. The table is empty apart from the "methods" column.

Chapter 4.7.1 describes how to set up a conditional table, chapter 4.7.2 describes the specification of the conditions and instructions and chapter 4.7.3 describes experimenting with conditional tables.

Description of the Menu Functions

- **Component**
  - *Clean code generation directory*

    Deletes all files in the code generation directory.
  - *Touch*

    Forced regeneration during the next code generation (*Flat* → the edited component, *Recursive* → the referenced components too).
  - *Generate Code*

    The code for one component is generated.

- *Compile*

  Compiles the generated code.

- *Open Experiment*

  An experiment is launched.

- *Default Project*

  Allows the editing of the default project which is used when experimenting with components
  (*Edit* → opens an editor for the default project,
  Resolve Globals → automatically generates a global element for every imported element of the component for which there is no exported element,
  Delete Unused Globals → löscht nicht genutzte globale Elemente).

- *Edit Layout*

  The layout editor opens.

- *Edit Data*

  The data editor for a component opens. Data of one component can be searched.

- *Edit Implementation*

  The implementation editor opens. Implementations of one component can be searched.

- *Edit Notes*

  Opens the notes editor – this is where notes on the component can be made.

- *Check Dependency*

  A check is carried out to see whether the allocation of the formal parameters to the model parameters is correct.

- *Export Data*

  Exports the dataset of a component.

- *Show Path*

  Shows the path of an included component.

- *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard
  (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path

Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

– *File Out Generated Code*

  Saving the generated code in the file system
  (*Flat* → the edited component,
  Recursive → the referenced components too).

– *View Generated Code*

  Generates the code for the component and displays it in a text editor.
  The text editor can be selected in the ASCET options window, "ASCII Editor" node (cf. page 60).

– *File Out Generic Code For External Make*

  Saves the generated code in a directory of your choice to be processed later with separate tools, e.g. in an external Make/Build process.

– *Exit*

  Exits the conditional table editor.

• **Element**
  (is also available as a context menu in the "Elements" area)

Note

*The **Add Item**, **Rename**, **Delete** and **Edit** functions are only available when ASCET-MD is installed.*

– *View → Collapse all* collapses the tree structure in the "Elements" area so that only the component itself can be seen.

– *View → Expand all* expands the tree structure completely so that the entire content of the component is displayed.

– *Add Item*

  Adds a component as a complex element.

– *Rename (*<F2>*)*

  Renames a selected diagram element.

– *Delete (*<DEL>*)*

  Deletes a selected diagram element.

– *Show Occurrences*

  Shows all occurrences of an element (only in the block diagram editor).

– *Edit*

  Edits the configuration of a selected element.

– *Edit Data*

  Edits the data of a selected element.

– *Edit Implementation*

  Edits the implementation of a selected element.

– *Set Cache Locking*

  These menu options are used in connection with ASCET-RP. They are described in the ASCET-RP user's guide.

– *Edit Component*

  Opens the specification editor for a selected added component.

– *Replace Component*

  Replaces a component by another component. The name of the old component is retained.

– *Show Path*

  Shows the path of a selected added component.

– *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard
  (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path
  Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

– *Notes*

  Opens the notes editor for a selected added component.

– *Edit Distribution / Edit Max Size*

  As conditional tables can only contain scalar basic elements, these menu functions are disabled.

– *Copy Elements (*<CTRL> + <C>*)*

  Copies one or more selected elements from the "Elements" list.

- *Paste Elements (<CTRL> + <V>)*

  Adds one or more copied elements into the "Elements" list.
- *File Out Data / File In Data*

  As conditional tables can only contain scalar basic elements, this menu function is disabled.
- *Export Data*

  Exports the dataset of an added component.

• **Column**
(is also available as a context menu in the table area)

- *Add*

  Adds a new column
  (*Condition* → in the condition area,
  Instruction → in the instruction area).
- *Delete*

  Removes the selected column (without deleting the relevant element).
- *Move Left*

  Moves the selected column to the left.
- *Move Right*

  Moves the selected column to the right.

• **Row**
(is also available as a context menu in the table area)

- *Add*

  Adds a new row
- *Delete*

  Deletes the selected row.
- *Move Up*

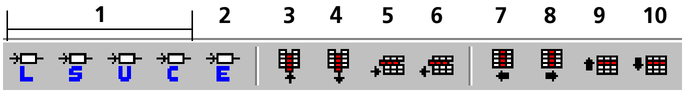
  Moves the selected row up.
- *Move Down*

  Moves the selected row down.

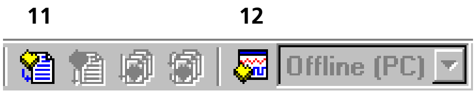
- **View**
  - *Show/Hide*

    Shows/hides different parts of the editor.
    (*Element List* → "Elements" list,
    Diagram List → "Diagrams" list,

Description of the Buttons



1. Variables (Logic, Signed Discrete, Unsigned Discrete, Continuous)
2. Enumeration
3. Add a Column
4. Delete a Column
5. Add a Row
6. Delete a Row
7. Move Column Left
8. Move Column Right
9. Move Row Up
10. Move Row Down



11. Generate Code
12. Open Experiment for selected Experiment Target

Unnumbered buttons are always disabled in the conditional table editor.

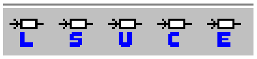
4.7.1 Setting Up a Conditional Table

By default, the conditional table contains no elements. The table is empty apart from the "methods" column. You have to add all the elements, columns and rows required. You cannot add columns until you have created at least one element; you cannot add rows until there is at least one column in the condition area.

To add elements:

Conditional tables can contain scalar basic elements, enumerations and classes as elements. The `trigger` method can contain arguments but no return value.

1. Basic elements, enumerations



- Click on one of the variables or enumeration buttons to create a basic element or an enumeration.

  A corresponding element is added to the "Elements" list.

- Enter a name for the element and press <RETURN>.

2. Complex elements

- Add a class as described on page 217.

Note

You cannot add special classes such as state machines, CT blocks, Boolean tables or conditional tables.

3. Arguments and method-local variables of the `trigger` method

- Select the `trigger` method from the "Diagrams" list.
- Select **Edit** from the context menu

or

- double-click the name of the method.

  The interface editor opens. As `trigger` cannot have a return value, the "Return" tab is missing.

- Add the required arguments as described on page 193.
- Add the required method-local variables as described on page 195.

  Method-local variables are only available within the conditional table.

You can edit the data and implementations of the elements in the usual way; for more details consult the sections 4.11 "Editing Data" and 4.12 "Editing Implementations".

To make the elements accessible outside the conditional table, you have to activate the get or set ports of the elements (if available) or declare the elements as global. There is no other way of accessing elements in a conditional table.

To make elements accessible for external communication:

- Select an element from the "Elements" list.
- Select **Element** → **Edit**

or

- select **Edit** from the context menu to open the element editor.

  Depending on the element selected, the appearance of the element editor may vary (see chapter 4.10.1). The "Scope" field as well as the options **Set()** and **Get()** are always available.

1. Using Get/Set Ports

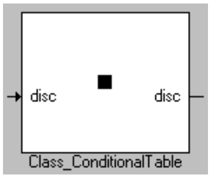
- Activate the **Set()** option to add an input for the element.

  The element can now be written from outside the conditional table.

- Activate the **Get()** option to add an output for the element.

  The element can now be read from outside the conditional table.

  The inputs and outputs created in this way can be seen as connections in the layout of the conditional table.



2. Using global elements

- Select the **Imported** option in the "Scope" field if the element was defined in another component/another project and is to be used in the conditional table.
- Select **Exported** if you define the element in the conditional table and want to use it in other components/projects.

Once you have added an element, the menu functions for creating columns for conditions and instructions are activated.

Note

The ***Add a Column*** *button can create columns in both the condition area and in the instruction area. Select an existing column to determine the area for the column to be created; this activates the button.*

A variable, an enumeration or an argument is assigned to every column when it is created. Complex elements cannot be assigned to the columns; they are only used in the specification of conditions or instructions.

To create columns in the condition area:

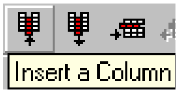
You can create up to 100 columns in the condition area.

- Select **Column** → **Add** → **Condition**.

  This function can be reached via the menu bar and the context menu of the table area

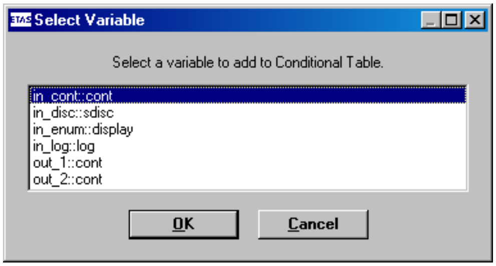
or

- activate the column next to which you want to add the new column.



- Click **Add a Column**.

  The "Select Variable" window opens. It contains a list of all existing variables which as yet have not been assigned to a column of the condition area.


Select Variable

Select a variable to add to Conditional Table.

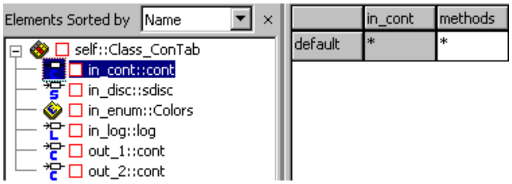
```
in_cont::cont
in_disc::sdisc
in_enum::display
in_log::log
out_1::cont
out_2::cont
```

- Select a variable from the list.

  You can assign each variable exactly once to a column of the condition area.

- Click **OK** to close the window.

  A column with the name of the selected variable is generated in the condition area. If the column created is the first of the entire table, the "Default" row is created at the same time.



```
self::Class_ConTab
  in_cont::cont
  in_disc::sdisc
  in_enum::Colors
  in_log::log
  out_1::cont
  out_2::cont
```

	in_cont	methods
default	*	*

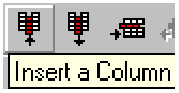
The new column is created on the right-hand side of the condition area if it was created with the menu function **Column** → **Add** → **Condition**. If you used the **Insert a Column** button, the new column is created to the right of the selected column.

To create columns in the instruction area:

You can create up to 100 columns in the instruction area.

- Select **Column** → **Add** → **Instruction** from the menu bar or the context menu

or



- activate the column next to which you want to add the new column.
- Click **Insert a Column**.

  The "Select Variable" window opens. It contains a list of all existing variables which as yet have not been assigned to a column of the instruction area.
- Select a variable from the list.

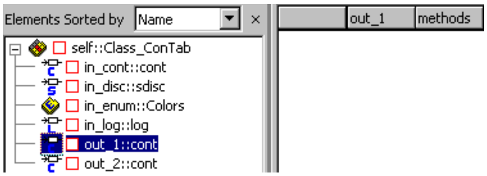
  You can assign each variable exactly once to a column of the instruction area.

Note

Arguments of the `trigger` *method* ***cannot*** *be assigned in the instruction area.*

- Click **OK** to close the window.

  A column with the name of the selected variable is generated in the instruction area. Even if the column created is the first of the entire table, *no* row is created.



The new column is created to the left of the "method" column if it was created with the menu function **Column** → **Add** → **Instruction**. If you used the **Insert a Column** button, the new column is created to the right of the selected column.

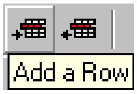
To create rows:

If at least one column has been created in the condition area, you can add rows. A conditional table can contain up to 100 rows; as the "default" row is already specified, you can add 99 rows.

- Select the row after which you want to add the new row.

- Select **Row** → **Add** from the menu bar or the context menu

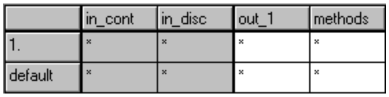
or



- click **Add a Row**.

  The new row is added after the one selected. If you have not selected a row or selected "default", the new row is added immediately before the "default" row.

  The elements of the row all initially contain an asterisk. For details on how to specify conditions and instructions, refer to chapter 4.7.2.



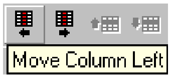
	in_cont	in_disc	out_1	methods
1.	*	*	*	*
default	*	*	*	*

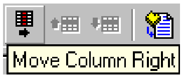
To move columns:

You can move columns but only within an area.

- Mark the column you want to move.
- Select **Column** → **Move Left** or **Column** → **Move Right** from the menu bar or the context menu

or





- click **Move Column Left** or **Move Column Right**.

  The column is moved one position to the left or right.

The first column of an area cannot be moved to the left; the last column of an area cannot be moved to the right.
The "methods" column cannot be moved; it is always the last column in the instruction area.

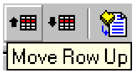
To move rows:

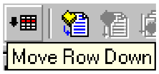
The further up the table a row is, the higher its priority. You can move the rows and thus influence the priority of the condition specified in it.

- Mark the row you want to move.

- Select **Row** → **Move Up** or **Row** → **Move Down** from the menu bar or the context menu

or





- click either **Move Row Up** or **Move Row Down**.

  The row is moved up or down a row.

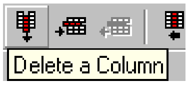
The first row cannot be moved up; the last numbered row cannot be moved down.
The "default" row cannot be moved; it is always the last row of the table.

You can remove columns, rows and elements from the table.

To remove columns:

- Select the column you want to remove.
- Select **Column** → **Delete** from the menu bar or the context menu

or



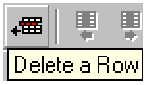
- click **Delete a Column**.

  The selected column is removed. The assigned element is not, however, removed but still remains in the "Elements" list and can be reassigned.

To delete rows:

- Select the row you want to remove.
- Select **Row** → **Delete** from the menu bar or the context menu

or



- click **Delete a Row**.

  The selected row is deleted.

To delete elements:

- Mark the element you want to delete in the "Elements" list.
- Select **Element** → **Delete**

or

- select **Delete** from the context menu

or

- press the <DEL> key.

  A confirmation window opens.

- Confirm deletion with **OK**.

  The element is deleted from the list. If it was assigned to a column, the column is also deleted.

Note

*If the deleted element was part of the specification of a condition or instruction, this specification is **not** deleted automatically. You must make any necessary changes manually.*

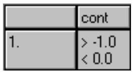
4.7.2 Specifying a Conditional Table

All cells of the newly created columns and rows initially contain an asterisk (*). You have to enter the real conditions and instructions manually.

Conditions: The cells in the condition area can contain conditions in the following format:

```
== b, <= b, >= b, < b, > b, != b, *
```

- b is either a value, an element or the name of a public method of a referenced class. Complex expressions formed from these in ESDL syntax are possible.
- The asterisk means that any value is possible; it is used when the column element is of no significance to the current row.
- Several conditions can be specified in one cell, separated by a line break. These are interpreted as being AND-operated.



	cont
1.	> -1.0 < 0.0

Note

*The cells in the condition area of the "defaults" row **cannot** be edited. They always contain an asterisk.*

Instructions: The cells in the instruction area can contain assignments in the following format:

```
= b, class.method(<variables>), *
```

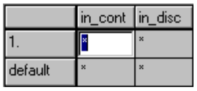
- b is a value, an element or the name of a public method of a referenced class. Complex expressions formed from these in ESDL syntax are possible.
- `Class.method(<variables>)` denotes a method call; this form of the instruction is only permissible in the "methods" column. All public methods of the referenced classes can be invoked.
- The asterisk means that no instruction (value assignment or method call) is defined in the relevant column.

You can only use elements and classes contained in the "Elements" list to specify conditions and instructions. Otherwise an error message is created during code generation.

Note

Syntax and semantics are not checked during the specification of conditions and instructions. Make sure your entries are correct, particularly when entering method names. Error messages are not created until code generation.

To enter a condition or an instruction:



	in_cont	in_disc
1.	*	*
default	*	*

- Double-click the cell you want to edit.

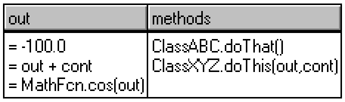
  The cell becomes an input box.
- Enter the condition or the instruction.

  Conditions and instructions are entered as pure character strings.

Note

No check is made to see whether different rows in the condition area contain identical conditions.

A cell in the instruction area can also contain several rows. Each row is interpreted as an individual assignment (see figure) or method call ("methods" column).



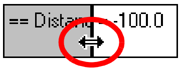
out	methods

= -100.0
= out + cont
= MathFcn.cos(out)
	ClassABC.doThat()
	ClassXYZ.doThis(out,cont)

- Click somewhere in the table area to accept the entry.

To alter column width:

If the width of a column is too small for the instructions or conditions contained in the cells, you can widen it.



- Move the mouse pointer to the right-hand vertical limitation of the column.

  The mouse pointer becomes a double arrow.

- Drag the column limitation to the left or right to change the width of the column.

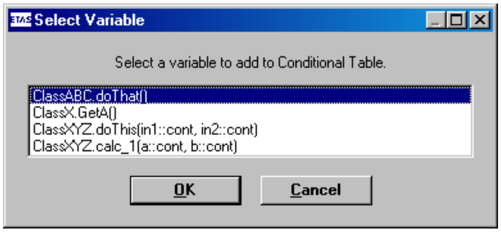
  When you exit the editor, the column width is reset to the default value.

To use methods in conditions/instructions:

There is a way of avoiding errors when entering method names.

- Mark the cell in which you want to use a method.
- Click the table area with the right-hand mouse button and select **Insert Method** from the context menu.

  The "Select Variable" window opens. It contains a list of all public methods of all referenced classes.



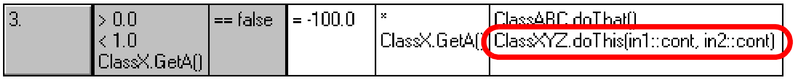
Select a variable to add to Conditional Table.

ClassABC.doThat()
ClassX.GetA()
ClassXYZ.doThis(in1::cont, in2::cont)
ClassXYZ.calc_1(a::cont, b::cont)

- Select the method you want to use.

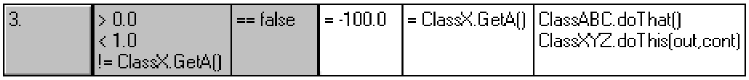
- Click **OK**.

  The name of the method is inserted in an individual row at the end of the cell.
  If the method has arguments, the names defined in the class are entered.



3.	> 0.0
	< 1.0
	ClassX.GetA()
		== false	= -100.0	*
				ClassX.GetA(
					ClassABC.doThat()
					ClassXYZ.doThis(in1::cont, in2::cont)

- You have to edit the inserted method call.
  - Enter existing variable names from your conditional table instead of the specified argument names.
  - Insert a comparing operator in the condition area (see page 366).
  - Add the assignment character = in the instruction area (apart from the "methods" column) and remove the asterisk if necessary.



3.	> 0.0
	< 1.0
	!= ClassX.GetA()
		== false	= -100.0	= ClassX.GetA()	ClassABC.doThat()
					ClassXYZ.doThis(out,cont)

4.7.3 Experimenting with Conditional Tables

Once you have specified the table completely, you can generate code and execute experiments as for other components (see chapter 4.1.10 "Analyzing Components"). Only the **Diagram** → **Analyze Diagram** function available for block diagrams and ESDL components is not available for conditional tables.

The following problems may occur during code generation:

- **Any incorrect entries** made when specifying the conditions and instructions result in error messages.
- If you have specified **identical conditions** in different rows of the condition area, all corresponding `If`...`Then`... rows are generated during code generation but only the first one is taken into consideration when the table is evaluated.
  No warning is displayed as there is no check of the content of the table.

- If you **delete, add or move enumerators** in an enumeration used by the conditional table, this may lead to inconsistencies in the conditional table. A relevant error message may be displayed during code generation.
- If you **delete** an **enumeration** used by the conditional table from the **database** , the enumeration is treated as an undefined element in code generation; a corresponding error message is displayed.

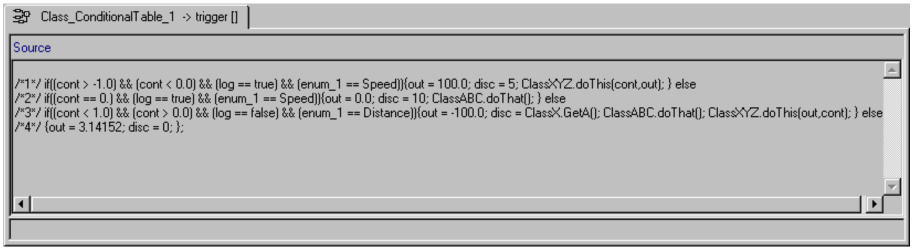
The conditional table can only be invoked via the `trigger` method. The following steps are executed every time the table is invoked:

1. The method arguments of the `trigger` method are processed and the values assigned to the elements.
2. A search takes place for a suitable condition. As soon as a suitable condition is found, the search is canceled, even if there are other suitable conditions specified further down.

   If there is no suitable condition, the default condition ("default" row) is selected.
3. The instructions belonging to the condition found are executed.

   As `trigger` cannot have a return value, the results of the instructions have to be transferred using Get Ports (see page 360). To ensure data consistency at all times, it is not recommended that you use global variables.

As with all ESDL components, the code is displayed for an offline experiment in the "Physical Experiment" window. This is the only point at which the user sees the ESDL code.



```
/*1*/ if((cont > -1.0) && (cont < 0.0) && (log == true) && (enum_1 == Speed)){out = 100.0; disc = 5; ClassXYZ.doThis(cont,out); } else
/*2*/ if((cont == 0.) && (log == true) && (enum_1 == Speed)){out = 0.0; disc = 10; ClassABC.doThat(); } else
/*3*/ if((cont < 1.0) && (cont > 0.0) && (log == false) && (enum_1 == Distance)){out = -100.0; disc = ClassX.GetA(); ClassABC.doThat(); ClassXYZ.doThis(out,cont); } else
/*4*/ {out = 3.14152; disc = 0; };
```

You can set up and execute the experiment as described in chapter 6.1 "The Experimentation Environment".

4.8 The Project Editor

A project in ASCET defines the functionality of an embedded software system and is used as the basis for generating code for the embedded system. This kind of specification can be executed on various experimental targets and is capable of running in real-time. Fixed point code for microcontrollers can be generated once an implementation transformation for the embedded software system has been defined. This fixed point code can then be run on microcontroller targets in fullpass or bypass experiments.

Alternatively, a project can be used to model a control loop consisting of a combination of continuous time blocks and modules. In this case the continuous time blocks are used to describe the process model, and the modules are used to describe the controller. This model can then be experimented with on an experimental target.

A project is developed in a modular manner, by first developing and testing the individual components of the system, and then combining them into a project. Developing a project involves the following steps:

- Selecting the modules that make up the system. The definitions of the components are referenced, i.e. if the definition of a component is changed, this change directly affects all the projects that use the component.
- Defining the overall control flow of the embedded software system by setting up the real-time operating system.
- Specifying the implementation transformation for generating code with fixed point arithmetic. In order to define this implementation, the transformation formula must be included in the project.

Like components, projects can have multiple data sets and implementations. Therefore when executable code is generated from the formal definition the appropriate variant has to be chosen. The current variant of a project is defined by selection of the data set (this is also possible when experimenting with components only) and of the implementation transformation.

Defining an implementation is only necessary for generating code with fixed point arithmetic. When only floating point arithmetic is needed (e.g. in a bypass environment) the implementation transformation need not be defined.

The code generation can be adjusted for a specific project by selecting the code variant and the target platform for which code is to be generated. Code can be generated for three types of arithmetic: floating point arithmetic, fixed point arithmetic and quantized floating point arithmetic. The latter is a simulation of fixed point arithmetic based on floating point arithmetic, where the effects of quantization can be studied, and the quantization and the value bounds can be changed interactively while executing the code.

To create a new project:

- In the Component Manager, select a folder for the new project.
- Select **Insert** → **Project**

or

- click on the **Insert Project** button.
- Type in the name for the new project and press <ENTER>.
- A new project is created.
- From the menu bar, select **Component** → **Edit Item**

or

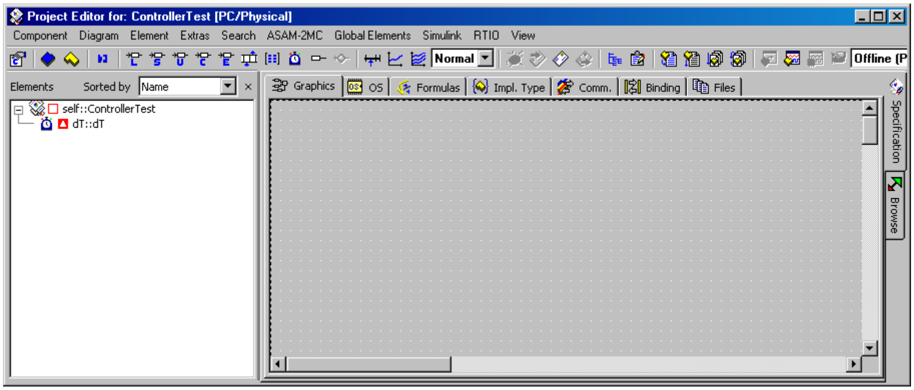
- press <RETURN>

or

- in the "1 Database" list, double-click on the project name.

  The project editor opens.

Like components, projects are stored in the database with the exception of the test projects generated by the specification editor. Individual projects are assembled, set up and experimented with in the project editor. All the operations described above are accessible from the project editor.


Project Editor for: ControllerTest [PC/Physical]
Component Diagram Element Extras Search ASAM-2MC Global Elements Simulink RTIO View
Normal
Offline (P
Elements Sorted by Name
self::ControllerTest
dT::dT
Graphics OS Formulas Impl. Type Comm. Binding Files
Specification
Browse

The "Elements" pane provides a tree-like view of the components making up the project. This list displays the modules referenced by the project and also the components they reference. It can be expanded and collapsed. The elements of a project used for communication between the modules and components are global in the context of the project.

4.8.1 Default Project for a Component

A test or default project can be created for each component created with a specification editor. However, the default project does not appear in the Component Manager, as it is created directly from the specification editor. Similarly, the default project is set up in the project editor and this also includes experimenting with the default project. Project editor functions and operations are described in the following chapters and sections.

To edit a default project:

- In the specification editor, select **Component** → **Default Project** → **Edit**.

  The project editor opens for the test project.

- In the specification editor, select **Component** → **Default Project** → **Resolve Globals**.

  A global element is created for each imported element in the component for which there is no exported element (see also "Defining Global Communication" on page 388).
  The project editor is not opened.

- In the specification editor, select **Component** → **Default Project** → **Delete Unused Globals**.

  Global elements not used int he component are deleted from the component's default project.

The offline experimentation environment available for components is embedded in the offline experimentation for the default project. Besides the offline experimentation environment, ASCET provides an online experimentation environment for projects. In the online experimentation environment the projects are executed in real-time with the behavior defined in the real-time operating system.

- **Component**
  - *Clean code generation directory*

    Deletes all files in the code generation directory.
  - *Touch*

    Forced regeneration during the next code generation
    (*Flat* → the edited component,
    Recursive → the referenced components also).
  - *Generate Code (<*CTRL*> + <F7>)*

    Generates the code for the project.
  - *Compile*

    Compiles the generated code.
  - *Link Only*

    Links the object files of a project to an executable hex file.
  - *Build (<F7>)*

    Generates, saves and builds executable code for the target selected. Results are stored in the database.
  - *Build from directory] (<*ALT*> + <F7>)*

    Generates, saves and builds executable code for the target selected. Results are written to a directory, but not stored in the database.
  - *Rebuild All (<*SHIFT*> + <F7>)*

    A complete regeneration of the entire project and all its components. This command produces the same result as **Component** → **Touch** → **Recursive** followed by **Component** → **Build**.
  - *Transfer Project*

    Transfers the project to the selected experiment.

    The button is only enabled when you have selected the target `ES1130, ES1135`, or Prototyping from the build options and the entry `INCA` or `INTECRIO` from combo box "Experiment Target".
  - *Open Experiment*

    Starts an experiment.

- *Flash Target*

  The code generated by ASCET is written to the flash memory of the experimental target instead of to the RAM. A startup routine for booting from the flash memory is integrated into the code. The target hardware will now execute the ASCET model after each reset.
- *Reconnect to Experiment*

  Restores the connection to the experiment running on the selected target.
- *Default Project*

  Allows editing of the default project used for experimenting with components. (Active in the component editors only.)
- *Edit Data*

  The project editor opens for the project. Search of project component data is possible.
- *Edit Implementation*

  The implementation editor opens for the project. Search of project component implementations is possible.
- *Edit Notes*

  Opens the Notes editor for notes about the component.
- *Edit Project Properties*

  Opens the "Project Properties" dialog window used to specify the project settings (see "Project Settings" on page 407).
- *Check Dependency*

  Checks whether the allocation of formal parameters to the model parameters is correct.
- *Export Data*

  Exports a component or project data set.
- *Show Path*

  Shows the path of an included component.
- *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard
  (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path

Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

- *File Out Generated Code*

  Saves the code generated in the file system.
  (*Flat* → the edited component,
  Recursive → the referenced components also)

- *View Generated Code*

  Generates the code for the project and displays it in a text editor. The text editor can be selected in the ASCET options window, "ASCII Editor" node (cf. page 60).

- *File Out Generic Code For External Make*

  Saves the code generated in any directory for processing at a later stage using external tools, e.g. an external Make/Build process.

- *Exit*

  Exits the project editor.

• **Diagram**

  - *Store To Cache*

    The component specification is saved in the cache. (It is not saved permanently in the database.)

  - *Analyze Diagram*

    Analyzes the current diagram.

  - *Load Diagram*

    Loads a diagram.

• **Element**

  - *View* → *Collapse all* used to collapse the tree structure in the "Elements" pane so that only the component is shown.

  - *View* → *Expand all* expands the tree structure so that the entire content of the component is visible.

  - *Add Item*

    Accepts a component into a project.

  - *Rename (*<F2>*)*

    Renames a component.

  - *Delete (*<DEL>*)*

    Deletes an instance of a component.

– *Show Occurrences*

  Displays all the occurrences of an element (block diagram editor only).

– *Edit*

  Edits an element configuration.

– *Edit Data*

  Edits element data.

– *Edit Implementation*

  Edits the implementation of an element.

– *Set Cache Locking*

  These menu options are used in connection with ASCET-RP. They are described in the ASCET-RP user's guide.

– *Edit Component*

  Opens the specification editor for an included component.

– *Replace Component*

  Replaces a component with another component. The name of the old component remains.

– *Show Path*

  Displays the path of an included component.

– *Copy Path to Clipboard*

  Copies the path of an included component to the clipboard (*Model Path* → model path,
  Model asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in a model as hyperlink,
  Database Path → database path
  Dabase asd:// link → path in the ASCET protocol format that allows opening/referencing an ASCET component in the database as hyperlink).

– *Notes*

  Opens the Notes editor for an included component.

– *Edit Distribution*

  Opens an editor for distribution (only in group tables).

– *Edit Max Size*

  Opens an editor where you can specify the maximum size of an array, any table or matrix.

  - *Copy Elements (<*CTRL*> + <*C*>)*

    Copies an element from the "Elements" list.
  - *Paste Elements (<*CTRL*> + <*V*>)*

    Pastes a copied element into the "Elements" list.
  - *File Out Data*

    Writes the data from an array or a table to a file.
  - *File In Data*

    Reads the data for an array or a table from a file.
  - *Export Data*

    Exports the data set from an included component.
- **Extras**
  - *Copy C-Code From*

    Copies C-code from another target.
  - *Global Replace Formula*

    Replaces a formula recursively.
  - *Update Implementations*

    Updates all the implementations of a project.
- **Search**

  Apart from a more limited search range, the commands function in the same way as in described in "Browsing the Database" on page 133.
  - *Component*

    Searches the project for a component.
  - *References to component*

    Searches the project for references on a given component.
  - *Declaration of method*

    Searches the project for the defining component of a given method/process.
  - *References to methods*

    Searches the project for a method or process.
  - *Declarations of element*

    Searches the project for the defining component of a given element.

- *References to element*

  Searches the project for a component using a given element.
- *Senders of message*

  Searches the project for modules that send a given message.
- *Receivers of message*

  Searches the project for modules that receive a given message.
- *Find/Replace ESDL or C code*

  Searches/replaces a string in included C code or ESDL components

• **ASAM-2MC**

- *Write*

  Creates application files.
- *Read Hex. File*

  Reads application files.

• **Global Elements**

- *Resolve Globals*

  This procedure automatically creates a global element for each imported element in a module for which there is no exported element.
- *Delete Unused Globals*

  Deletes unused global elements.

The following menu options are only available when certain tabs have been selected.

"Graphics" Tab

• **View**

- *Show/Hide → Element List*

  Shows/hides the "Elements" list.
- *Redraw*

  Redraws the diagram.
- *Rebuild Connections*

  Smoothes fragmented connections that were created during automatic conversion of operator implementations (see page 508).
- *Undo (<CTRL> + <Z>)*

  Reverses the most recent action.

- *Redo (<CTRL> + <Y>)*

  Reverses an undo command.
- *Show Hierarchy Path*

  Displays the highest hierarchy path level.
- *Parent Component*

  Opens an editor for the including component. This option is only available when an included component is being edited.
- *Parent Hierarchy Level*

  Displays the hierarchy path.
- *Page frame Portrait*

  Displays the diagram in portrait format.
- *Page frame Landscape*

  Displays the diagram in landscape format.
- *Grid*

  Modifies the grid in the drawing area.
- *Print Diagram*

  Prints the content of the "Graphics" tab (see page 265).
- *Save as Postscript/ Bitmap/ RTF/ GIF*

  Saves the diagram in the format specified.

"OS" Tab

- **Operating System**
  - *Copy From Target*

    Converts a project from a selected target to the current target.
  - *Copy To Target*

    Converts the project to a selected target.
- **Application Mode**

Note

The menu functions are also available as context menu in the "Application" field.

  - *Add*

    Creates a new operating mode.

  - *Rename*

    Renames an operating mode.
  - *Delete*

    Deletes an operating mode.
  - *As Start Mode*

    Defines the operating mode as start mode.
  - *As Default CT-Mode*

    Sets the operating mode as standard mode for CT blocks.
  - *Assign*

    Assigns an operating mode to a task.
  - *Show inTasks*

    Selects all the tasks to which the operating mode selected is assigned.
  - *Show Init Tasks*

    Selects all the initialization tasks to which the operating mode is assigned.

- **Process**

Note

The menu functions are also available as context menu in the "Processes" pane.

  - *Assign*

    Assigns a process to a task.
  - Show in Tasks

    Shows all occurrences of a process in the "Tasks" pane.

- **Task**

Note

The menu functions are also available as context menu in the "Tasks" pane.

  - *Add*

    Creates a task.

- *Rename*

  Renames a task.
- *Delete*

  Deletes a task.
- *Move Up*

  Moves the process upwards within the task.
- *Move Down*

  Moves the process downwards within the task.
- *Deassign Application Modes*

  Removes an operating mode from the current task.
- *Deassign Processes*

  Removes a process from the current task.
- *Delete Undefined Processes*

  Deletes undefined processes from the current task.
- *Open Module*

  Opens the module which contains the selected process.
- *Show in Processes*

  Shows all occurrences of a process in the "Processes" pane.
- *Show in Application Modes*

  Selects all operating modes belonging to a task.
- *Set Cache Locking*

  These menu options are used in connection with ASCET-RP. They are described in the ASCET-RP user's guide.

"Formulas" Tab

- **Global Formulas**

Note

The menu functions are also available as context menu in the tab.

- *Add*

  Adds a formula.
- *Rename*

  Modifies the formula name.

- *Delete*

  Deletes the formula.
- *Edit*

  Opens the Formula editor.
- *File Out*

  Stores formulas in a file
  (*All* → all the formulas,
  Selected → selected formulas).
- *File In*

  Adds formulas from a file
  (*All* → all the formulas in the file,
  Select → formulas selected in the file).
- *Select all* (<CTRL> + <A>)

  Selects all formulas.

"*Impl. Type*" *Tab*

- **Global Implementations**

Note

The menu functions are also available as context menu in the tab.

- *Add*

  Add an implementation type
  (*cont* → model type `cont`,
  sdisc → model type `sdisc`,
  udisc → model type `udisc`)
- *Rename*

  Changes the name of an implementation type.
- *Delete*

  Deletes the selected implementation types.
- *Edit*

  Opens the editor for implementation types.
- *Copy implementation*

  Copies the current settings of the selected implementation type to the database clipboard.

- *Paste implementation*

  Pastes the current settings from the database clipboard to the selected implementation type.
- *File Out*

  Files out implementation types into an XML file
  (*All* → all implementation types,
  Selected → selected implementation types).
- *File In*

  Files in implementation types from an XML file
  (*All* → all implementation types in the file,
  Select → selected implementation types in the file).
- *Select all (<CTRL> + <A>)*

  Selects all implementation types.

"Files" Tab

- **Project Files**

Note

The menu functions are also available as context menu in the tab.

- *View*

  Shows the project file with the associated application.
- *Edit*

  Edits the project file with the associated application.
- *Add*

  Adds an external file.
- *Delete (<DEL>)*

  Deletes a file.
- *Update*

  Updates selected project files.
- *Write Selected File(s)*

  Writes selected project files to the default directory.
- *Write Selected File(s) to*

  Writes selected project files to the selected directory.

- *Write All Files*

  Writes all the project files belonging to the project to the default directory.
- *Replace With Backup Contents*

  Replaces selected project files with backup versions from the database.
- *Show Default Path*

  Shows the path of the default directory.
- *Explore Default Path*

  Opens the Explorer with the path of the default directory.
- *Copy From Target*

  Converts a project from a selected target to the current target.
- *Copy To Target*

  Converts the project to a selected target.
- *Select all (<CTRL> + <A>)*

  Selects all project files.

4.8.3 Specifying a Project

Modules and continuous time blocks are included by adding them to the "Elements" pane of the project editor as in other specification editors.

To include a component in a project:

- In the project editor, select **Element** → **Add Item**.

  The "Select Item…" dialog opens.
- From the "1 Database" list in the "Select Items..." window, select the component you want.
- Click **OK.**

  The component is added to the "Elements" pane of the project.

Each module used in a project has an instance name. As in the specification editors, each component is given a default name when it is included in a project. This name only applies only to the instance of the module in the project and has no influence on the original module.

To rename a component:

- In the "Elements" pane, select the module you want to rename.
- Select **Element** → **Rename**.
- Type in the new name and press <RETURN>.

To delete an instance of a component:

- In the "Elements" pane, select the component you want to delete.
- Select **Element** → **Delete**.

  The module instance is deleted. This has no effect on the component in the database.

It is possible to open the appropriate editor and edit any component that is part of a project from within the project editor. Editing a component from within a project has the same effect as editing it from the Component Manager, namely that the functional description is changed which affects all the instances of the component.

When opening a specification editor from within the project editor, the current project settings become active in the specification editor. If, for instance, the current project is set to fixed point code generation, the component editor will also be set to fixed point code generation. Only floating point code generation is available if the specification editor is started from the Component Manager.

To edit a component in a project:

- In the "Elements" pane of the project editor, select the component you want to edit.
- Double-click on the component name

or

- select **Element** → **Edit Component**

or

- press <RETURN>.

  The editor opens for the component you want to edit.

It is possible to edit the data and implementation of elements within a project, and to read and write data to and from them. All the commands work in the same way as in the block diagram editor.

You can attach notes to the project or to the included components. When documentation is generated automatically, the notes are included.

To edit the notes for a project:

- If you want to edit the notes of the project, select **Component** → **Notes**.

Or

- In the "Elements" pane or in the "Graphics" tab, select the included component whose notes you want to edit.
- Select **Element** → **Notes**.

  The notes editor opens for the database item selected. For details see "Notes" on page 693.

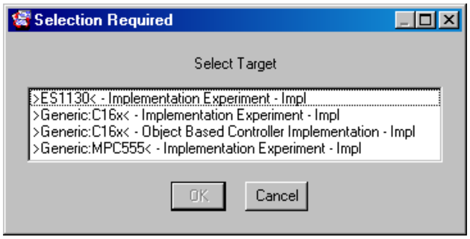
When editing a project that was written for a different target, the C code and operating system (see page 392) settings may be copied from the other target, experiment type, or implementation.

To copy the C code for an entire project:

To copy the C code for all classes and modules of a project from another target, experiment type, or implementation, proceed as follows.

- In the project editor, select the appropriate target and code generation options for your target, as described in chapter 4.8.8.
- Select **Extras** → **Copy C-Code From**.

  The "Selection Required" window opens.



Selection Required

Select Target

>ES1130< - Implementation Experiment - Impl
>Generic:C16x< - Implementation Experiment - Impl
>Generic:C16x< - Object Based Controller Implementation - Impl
>Generic:MPC555< - Implementation Experiment - Impl

- Select the target you want to copy the code from, and click **OK**.

  The target code is copied to your target.

4.8.4 Defining Global Communication

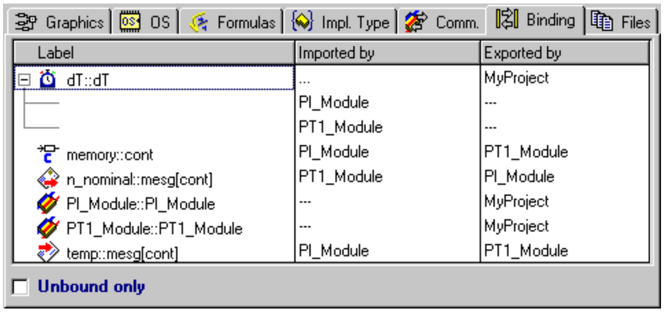
In components, the flow of data is organized through the interface elements of the components, i.e. by defining inputs and arguments and reading outputs and return values. In a project, communication is defined via imported and exported elements; these function like global variables. These elements are mapped onto each other according to their names, which have to be identical for two elements to be mapped. Therefore it is necessary to assign the names in the modules so that they will match in the project.

An element can be exported either by a module or by the project itself. Each exported element can be imported several times by other modules, but an element can only be exported once. If two modules within a project contain an exported element of the same name, an error message is displayed.

Elements can also be created in the project. They are then linked to elements with the same name in the modules. Binding is always automatic, i.e. the user can only influence the binding of imported and exported elements by assigning matching names to them.

To view the binding of variables in a project:

- In the project editor, click on the "Binding" tab.



Label	Imported by	Exported by
dT::dT	...	MyProject
	PI_Module	---
	PT1_Module	---
memory::cont	PI_Module	PT1_Module
n_nominal::mesg[cont]	PT1_Module	PI_Module
PI_Module::PI_Module	---	MyProject
PT1_Module::PT1_Module	---	MyProject
temp::mesg[cont]	PI_Module	PT1_Module

☐ **Unbound only**

The variable bindings for the project are shown.

- You can tick the **Unbound Only** check box at the bottom of the dialog tab to filter the list.

There are two included components and four global elements in the example above:

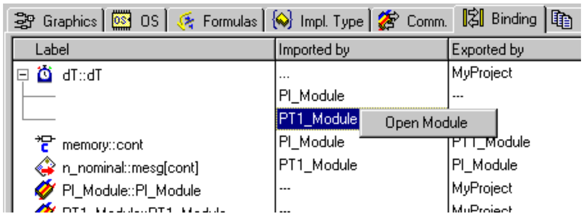
- The variable `dT` exported by the `MyProject` project. This variable is always created by default in the project and imported by all the referenced modules.
- The two included modules `PI_Module` and `PT1_Module`.
- The `n_nominal` message exported by the `PI_Module`. A Send message is defined with this name. It is imported by the module `PT1_Module`, where a receive message with the same name is defined.
- The message `temp`, which is exported from `PT1_Module` and imported by `PI_Module`.
- The variable `memory`, which is exported by `PT1_Module` and imported by `PI_Module`.

By default, send messages and send&receive messages are defined as exported elements, whereas receive messages are defined as imported elements. It is possible to change these assignments. Variables always have to be declared explicitly as either imported or exported elements. If an element is imported by more than one module this is indicated by a hierarchical display, as in the case of the `dT` variable in the example above.

If you find that an element is erroneously exported more than once, you can open the exporting component from the "Binding" tab.

To open a component from the "Binding" tab:

- In the "Imported by" or "Exported by" column, right-click on a component and select **Open Module** from the context menu.



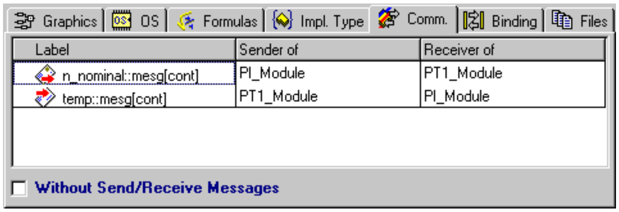
Label	Imported by	Exported by
dT::dT	...	MyProject
	PI_Module	---
	PT1_Module	
memory::cont	PI_Module	PT1_Module
n_nominal::mesg[cont]	PT1_Module	PI_Module
PI_Module::PI_Module	---	MyProject

Or

- Double-click on the component you want to edit.

  The editor for the selected component opens.

Click the "Comm" tab for a different view of the element mappings showing the messages defined within the project only. This shows the implicit data flow through send and receive messages.



Label	Sender of	Receiver of
n_nominal::mesg[cont]	PI_Module	PT1_Module
temp::mesg[cont]	PT1_Module	PI_Module

☐ Without Send/Receive Messages

This view only shows the messages and in which modules they are defined as send or receive messages. Activate the field at the bottom of the Dialog tab to filter this view.

A global element is created automatically for each imported element in a module for which there is no exported element. These two elements are then bound to each other.

To define global elements in a project:

- Select **Global Elements** → **Resolve Globals** to resolve the global elements.

Note

This command must be executed before an experiment can be started with this project.

Global elements that are created and not used in the project can be removed with the **Delete Unused Globals** command.

To delete unused global elements:

- Select **Global Elements** → **Delete Unused Globals** to delete all the unused global elements.

4.8.5 Defining the Scheduling in the OS Editor

The scheduling of processes is the second major operation performed in the project editor. The operating system does not activate individual processes directly. Instead, processes are grouped together by activation mode into sequences called tasks. These tasks are activated by the operating system and on activation the sequence of processes assigned to the task is executed in the given order.

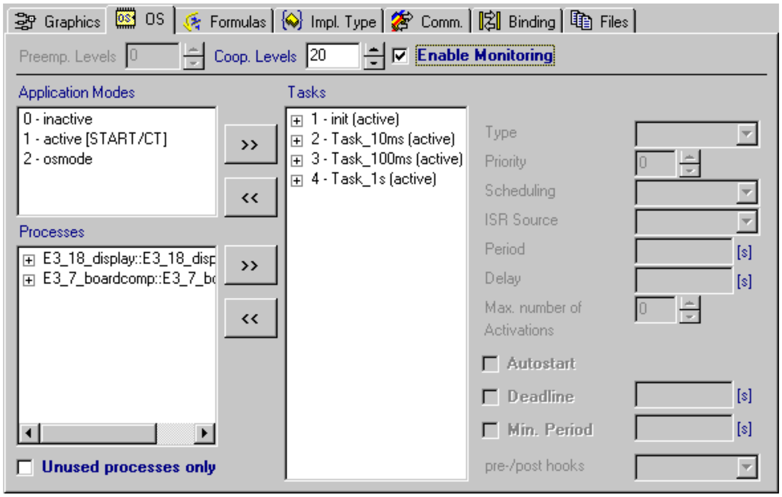
Note

All the settings in the operating system editor must be specified separately for each target. However, the settings can be copied between targets.

Operating System Settings

To set up the operating system:

- Click on the "OS" tab in the project editor to open the operating system editor



Preemp. Levels 0 Coop. Levels 20 ☑ Enable Monitoring

Application Modes
0 - inactive
1 - active [START/CT]
2 - osmode

Tasks
1 - init (active)
2 - Task_10ms (active)
3 - Task_100ms (active)
4 - Task_1s (active)

Processes
E3_18_display::E3_18_disp
E3_7_boardcomp::E3_7_bo

☐ Unused processes only

Type
Priority 0
Scheduling
ISR Source
Period [s]
Delay [s]
Max. number of Activations 0
☐ Autostart
☐ Deadline [s]
☐ Min. Period [s]
pre-/post hooks

- In the "Preemp. Levels" and "Coop. Levels" fields, set the number of co-operative and pre-emptive levels for the project.

  The number available depends on the implementation of the real-time operating system on the current target. If the target is a PC, for instance, there are no pre-emptive levels available, because pre-emptive multitasking is not possible in the PC implementation of the operating system.

- When you want to use the debugging function, activate the **Enable Monitoring** option.

The settings of the operating system for the current target can be copied to another target, or the settings from another target can be copied to the current target. Switching between targets is discussed in "Project Settings" on page 407.

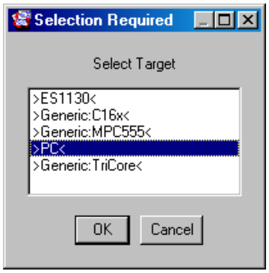
To copy operating system settings:

- Select **Operating System** → **Copy To Target**.

Or

- Select **Operating System** → **Copy From Target**.

  The "Selection Required" dialog box opens.



>ES1130<
>Generic:C16x<
>Generic:MPC555<
>PC<
>Generic:TriCore<

- Select a target and click **OK**.

  The settings are either copied from the current to the selected target, or the other way round, depending on which command was chosen.

Tasks and Processes

This section describes how to set up an operating system schedule using the operating system editor.

To create a task:

- Select **Task** → **Add**.

  A new task is created.

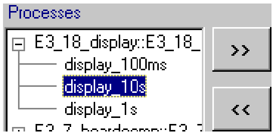
- In the "Tasks" pane, type in a name for the new task and press <ENTER>.

Once you have created the tasks required, you can add processes to each task.

To assign a process to a task:

- In the "Tasks" pane, select the task to which you want to assign a process.
- In the "Processes" pane, select the process you want to assign to the selected task.
- Select **Process** → **Assign**

or



- click on the **>>** button

or

- select **Assign** from the context menu

or

- drag the process from the "Processes" pane onto the desired task using the mouse.

  The process is assigned to the task and will be executed each time the task is scheduled.

To deassign a process from a task:

To deassign a process, proceed as follows.

- In the "Tasks" pane, select the process you want to deassign.
- Select **Task** → **Deassign Processes**

or

- click on the **<<** button (see figure above)

or

- select **Deassign Processes** from the context menu

or

- drag the process from the "Tasks" pane into the "Processes" pane using the mouse.

  The process is removed from the current task.

The order of processes within a task determines the order in which the processes are executed.

To shift a process in a task:

To shift a process in a task, proceed as follows.

- In the "Task" pane, select a process.
- Select **Task** → **Move Up** or **Task** → **Move Down** to specify the position of a process in the task.

To open a component from the "Tasks" pane:

If you want to edit a process, you can open the corresponding component from the "Tasks" pane.

- In the "Tasks" pane, select the process you want to edit.
- From the menu bar, select **Tasks** → **Open Module**

or

- select **Open Module** from the context menu

or

- Double-click on the process.

  The corresponding component editor opens.

To check assignments of application modes, tasks and processes:

1. To see the origin of a process assigned to a task proceed as follows:
   - In the "Tasks" pane, select a process.
   - Right-click on the process and select **Show in Processes** from the context menu

   or

   - select **Tasks** → **Show in Processes**.

     In the "Processes" pane, the selected process is highlighted.
2. To see the task a process is assigned to proceed as follows:
   - In the "Processes" pane, select a process.
   - Right-click on the process and select **Show in Tasks** from the context menu

   or

   - select **Process** → **Show in Tasks**.

     In the "Tasks" pane, the selected process is highlighted.

3. To see which application mode was assigned to a task, proceed as follows:

   - In the "Tasks" pane, select a task.
   - Right-click on the task and select **Show in Application Modes** from the context menu

   or

   - select **Tasks** → **Show in Application Modes**.

     In the "Application Mode" pane, the assigned application mode is highlighted.

4. To see which tasks are assigned to an application mode, proceed as follows:

   - In the "Application Mode" pane, select an application mode.
   - Right-click on the application mode and select **Show in Tasks** from the context menu

   or

   - select **Application Mode** → **Show in Tasks**.

     In the "Tasks" pane, the assigned tasks are highlighted.

Basic Task Settings:

The configuration of a task determines its priority among other tasks, its trigger mode and other details relating to task activation and execution.

To set up a task (basic settings):

1. Trigger Mode

   The following trigger modes are available:

   – *Alarm* tasks are triggered once at the beginning of every period defined in the "Period" field.

   – *Init* tasks are triggered only once, on start-up of the application mode they are assigned to.

   – *Software* tasks are triggered by operating system commands.

   – *Interrupt* tasks are triggered by hardware events. The hardware events available depend on the target. If, e.g., the target is a transputer board, an *event* task could be triggered by data arriving on channel 0.

– *Timetable* tasks (micro-controller targets with ERCOS^{EK} support only) are alarm tasks written into a timetable. Thus, runtime can be saved (at the price of enhanced memory requirement).

- From the "Type" combo box, select a trigger mode.

2. Scheduling

- In the "Scheduling" combo box, assign one of three possible scheduling modes to the task.

  This option is not available for *init* and *interrupt* tasks.

`cooperative`	If you select *cooperative*, the task will interrupt a running task of lower priority only after the currently running process is finished.
`preemptive`	If you select *preemptive*, the task will interrupt a running task with lower priority immediately. The currently running process is interrupted and will be resumed after the interrupting task is finished.
`non-preemptable`	Neither the current process nor the running *non-preemptable* task are interrupted if a task with a higher priority is activated.

Since, in ERCOS^{EK}, all preemptive Tasks have a higher priority than cooperative tasks, the effective interrupt behavior is as follows:

– Cooperative tasks never interrupt preemtive tasks.

– Cooperative tasks interrupt each other after the currently running proccess is finished (or even the task if the task to be interrupted has only one process assigned).

– Preemtive tasks always interrupt immediately.

Details on the scheduling modes are given in chapter 1.1.1 "Scheduling" of the ASCET reference guide.

Note

With some microcontrollers (e.g., MPC555), the usage of non-preemptable tasks is subject to certain conditions (see the respective ASCET-SE reference guide). When these conditions are neglected, corresponding error messages are generated by the code generation. ASCET-SE for RTA-OSEK offers a different selection of scheduling modes. See the ASCET-SE user's guide for details.

3. Priority

   If more than one task is scheduled for execution, their activation is determined by their priority.

   If, for example, several *alarm* tasks have been scheduled for simultaneous triggering, the task with the highest priority is triggered first. Tasks are interrupted if a task with a higher priority than the currently running task is activated. The general priority scheme is presented in chapter 1.1.2 "Tasks" of the ASCET reference guide.

   - Assign a priority to the task by entering a figure into the "Priority" field.

   Note

   Non-preemptable and preemptive tasks share a common priority region. Cooperative tasks have their own priority region.

   This option is not available for *init* tasks.

4. Trigger

   This setting determines which event acts as a trigger for an *interrupt* task. The combo box is unavailable for all other tasks.
   The options available in the combo box depend on the current target.

   - Select the trigger from the "ISR Source" combo box.

5. Others

   The period value determines how often the task is activated. If, for instance, the period value is 0.01, the task will be activated once every hundredth of a second.

   - Type a period value in seconds into the "Period(s)" field.

     This setting is only available for *alarm* and *timetable* tasks.

   When a *Delay* is specified, the respective task will be activated for the first time after the set delay time. If the delay is 0, the task will be activated once the program starts, and then at the beginning of every period.

   - Type a delay value in seconds into the "Delay" field.

     This setting is only available for *alarm* and *timetable* tasks.

The *Max. number of Activations* value determines how many times a task can be activated in parallel. If a task is re-activated before it has finished the execution following its previous activation, it is double activated. To save system resources, the maximum number of concurrent activations for each task can be limited.

- Adjust the value in the "Maximum No. of Activations" field.

  This setting is available for *alarm*, *timetable*, and *software* tasks.

Setting Up Hook Routines:

Note

This does not apply to ASCET-SE for RTA-OSEK.

You can specify the generation of hook routines for debugging purposes for each task.

To set up hook routines for a task:

- Make sure that the **Enable Monitoring** option is activated.
- From the "pre-/post hooks" combo box, select one of the following options.

`none`	no debug functions
`monitoring`	complete debug functions

Depending on the settings, suitable data structures are created during code generation, and the appropriate ERCOS^{EK} libraries are included. For this purpose, ASCET generates the make variable `E_HOOKS`. The following table shows the correlations between the settings in the OS editor and the `E_HOOKS` variable:

Enable Monitoring	pre-/post hooks	generated value for E_HOOKS	dT available in the Code
`yes`	"monitoring" for at least one task	`MONITORING`	yes, additional debug information
`no`	"monitoring" for at least one task	`DTONLY`	yes
any	"none" for all tasks	`DTONLY`	no

For simulations with experimental targets, only the `monitoring` option is important; it controls the generation of monitoring variables (s. page 400). The options come fully into operation only in connection with a micro-controller target, and are therefore described in the ASCET-SE user’s guide.

OSEK-Specific Settings:

In addition, you can set up OSEK-specific settings. Since these settings are relevant only in connection with a micro-controller target, please refer to the ASCET-SE V5.2 user’s guide for further information.

To set Deadline and Min. Period options:

- Activiate the **Deadline** option to make sure that two consecutive task activations do not exceed a certain time difference.

  The associated input field becomes available.
- Enter the maximum difference in seconds between two activations.

  The **Deadline** option is available for *alarm*, *timetable*, and *software* tasks.
- Activiate the **Min. Period**option to make sure that two consecutive task activations do not fall below a certain time difference.

  The associated input field becomes available.
- Enter the minimum difference in seconds between two activations.

  The **Min. Period** option is available only for *interrupt* tasks.

To set the Autostart option:

The **Autostart** option is available for *software* tasks and *alarm* tasks. It has a different meaning for either type.

- In the "Tasks" pane, select a *software* task.
- Activate the **Autostart** option.

  Thus, the software task is executed once at the initialization of the associated application mode.

Or

- In the "Tasks" pane, select an *alarm* task.

- Activate the **Autostart** option.

  You have thus determined that the timer for the selected task is started at the initialization of the associated application mode.

The Monitoring Option

The monitoring option provides three variables for every task to be used for monitoring and debugging purposes during experiments with projects, as well as one further variable for monitoring of all tasks. The variables are created automatically when the **Enable Monitoring** option is active, and when `monitoring` was selected from the "pre-/post hooks" combo box for the task during task setup. This has to be done individually for each task for which monitoring variables are to be generated. The following three task-specific monitoring variables are generated:

`cycleStartTime_<taskname>`	shows the point in time in seconds the task was last activated.
`cycleTime_<taskname>`	shows the time in seconds required for executing the task.
`dT_<taskname>`	shows the time difference between the previous and the current activation of the task in seconds.

In addition, the following variable is created once for all tasks:

`runtime_violation`	shows the number of times the task has not met its schedule.

To activate the monitoring option:

- In the "Tasks" list, select one or more tasks.
- Activate the **Monitoring** option in the lower right corner of the tab.

  The monitoring variables for each task are generated during the next code generation. You can show them in the measure windows or write them to the data logger (offline experiment only, cf. "The Data Logger" on page 597).

Application Modes

The operating system can have several application modes, each of which has its own set of tasks assigned to it. This means that different operating conditions on the controlled system can be modeled, e.g. an engine can have separate operating modes for the warm-up phase and for normal operation.

To create an application mode:

- Select **Application Mode** → **Add**.

  A new application mode is created and shown in the "Application Modes" field.
- Enter a name for the application mode and press <ENTER>.
- If the operating mode is to be the start mode, select **Application Mode** → **As Start Mode**.

  The application mode declared the start mode is the one that the system is in at start-up. Operating system commands are used to switch between application modes.

To assign application modes to a task:

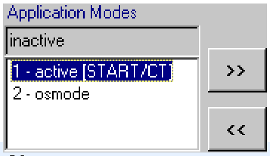
- Select one or more operating modes in the "Application Modes" pane.

  You can select several application modes by clicking on them while pressing the < CTRL> key.
- Select the task you want to assign to the application mode(s) in the "Tasks" pane.
- Select **Application Mode** → **Assign**

or

- select the desired application modes and drag them from the "Application Modes" pane onto the desired Task using the mouse.

  A task can have several application modes; it is scheduled according to its task settings whenever an application mode it is assigned to is active.



- Select **Task** → **Deassign Application Modes** to deassign the application mode.

  Instead of the menu functions, you can use the buttons shown here.

Note

Switching from one application mode to another is done via an operating system service call. For details, see the API description in the ERCOS^{EK} manual.

4.8.6 Administration of External Project Files

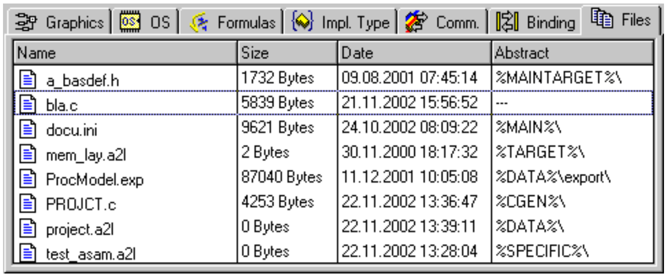
As a rule, there are more project files belonging to a project. These can be, for example, protocols, correspondence documents or C files. So that these external files can be assigned uniquely to the project, they are managed in the project editor via the "Files" tab.

The "Files" tab contains a table with name, size (in byte), and creation date of the files. The file paths are stored as absolute paths; relative paths are used only if a file is located in one of six preset directories. In that case, the "Abstract" column contains one of the following key words.

Key Word	Path
`%MAIN%`	`ETAS\Ascet5.2` (ASCET installation directory)
`%CGEN%`	`ETAS\Ascet5.2\cgen` (target directory of code generation)
`%MAINTARGET%`	`ETAS\Ascet5.2\target` (main target directory)
`%TARGET%`	`ETAS\Ascet5.2\target\<target name>` (target directory for current target; if, e.g., the PC target is selected, `TARGET` means `ETAS\Ascet5.2\target\PC`.)
`%DATA%`	`ETASData\Ascet5.2` (product data directory)
`%SPECIFIC%`	user-defined path, to be set in the options window (cf. page 49)

To add an external project file:

- In the project editor, click on the "Project Files" tab.



Name	Size	Date	Abstract
a_basdef.h	1732 Bytes	09.08.2001 07:45:14	%MAINTARGET%\
bla.c	5839 Bytes	21.11.2002 15:56:52	---
docu.ini	9621 Bytes	24.10.2002 08:09:22	%MAIN%\
mem_lay.a2l	2 Bytes	30.11.2000 18:17:32	%TARGET%\
ProcModel.exp	87040 Bytes	11.12.2001 10:05:08	%DATA%\export\
PROJCT.c	4253 Bytes	22.11.2002 13:36:47	%CGEN%\
project.a2l	0 Bytes	22.11.2002 13:39:11	%DATA%\
test_asam.a2l	0 Bytes	22.11.2002 13:28:04	%SPECIFIC%\

This tab displays all the associated project files.

- Select **Project Files** → **Add**.

  A file prompter dialog box is displayed.

- Select a file and press <ENTER>.

  The selected file is linked to the project editor.

To view a project file:

You can view a project file from the "Files" tab.

- In the "Files" tab, select the project file you want to view.
- From the menu bar, select **Project Files** → **View**

or

- select **View** from the context menu.

  The project file is opened in the associated application.
  If no application is associated with the file type, the command has no effect.

To delete a project file in the project editor:

- Select the file you want to delete in the project editor.

- Select **Project Files** → **Delete**.

  The file is deleted from the project (but not from the hard disk).

To write project files:

- Select one or more files in the project editor.
- Select **Project Files** → **Write Selected File(s)** to write the files to their standard directories.

Or

- Select **Project Files** → **Write Selected File(s) to** to write the files to a selected directory.

Or

- Select **Project Files** → **Write All Files** to write all the files to their default directories.
- If you want to know where the standard directory of a project file is located, select **Project Files** → **Show Default Path** to display the path.
- Select **Project Files** → **Explore Default Path** to open the standard directory of the selected file in the Explorer.

Note

***Project Files** → **Show Default Path** and **Project Files** → **Explore Default Path** are not available when more than one project files are selected..*

Update the project file:

If you have changed project file versions stored on disk, you can update them in the project as follows.

- Select one or more files in the project editor.
- Select **Project Files** → **Update**.

  The project file versions stored in the database are replaced with the versions stored on disk.

The database always keeps older versions as backup. If you need the backup version, proceed as follows.

- Select one or more files in the project editor.

- Select **Project Files** → **Replace With Backup Content**.

  The actual versions of the selected project files are replaced with their backup versions.

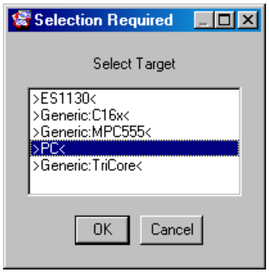
To copy the project file:

- Select **Project Files** → **Copy From Target**.

Or

- Select **Project Files** → **Copy To Target**.

  The "Selection Required" dialog opens.



```
>ES1130<
>Generic:C16x<
>Generic:MPC555<
>PC<
>Generic:TriCore<
```

- Select a target and click **OK**.

  The file is either copied from the current to the selected target, or the other way round, depending on which command was chosen.

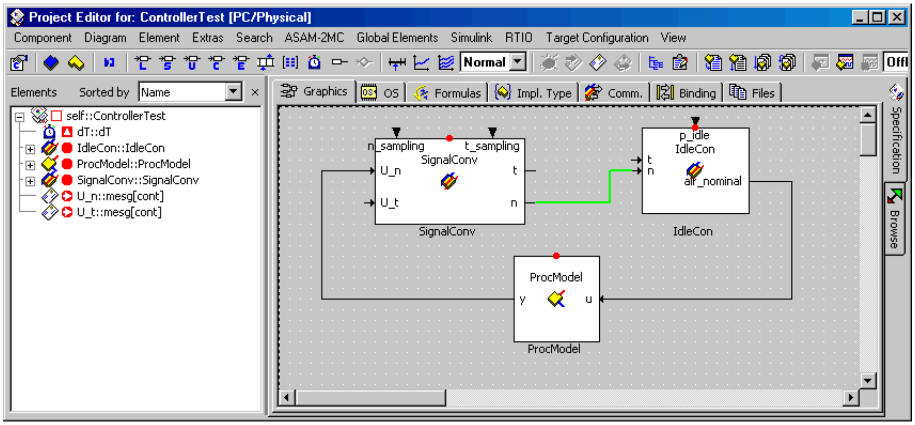
4.8.7 Hybrid Projects

Hybrid projects are projects that contain both continuous time blocks (CT blocks) and modules. CT blocks implement process models, i.e. mathematical models of the processes that are controlled by embedded control systems.

To set up a hybrid project:

- Set up the modules of the project as described earlier in this chapter.
- Include the continuous time blocks you want.
- Click on the "Graphics" tab of the project editor.
- Drag the continuous time blocks and modules you want to include in the hybrid project into the drawing area.

- Connect the CT blocks and the modules in the same way as elements in block diagrams.



You can change the layout of the components as described in section "Layout of Included Components".

The communication between CT blocks and between continuous time blocks and modules is formed by graphical connections like those in block diagrams. In the current version of ASCET you can also connect modules in this way. The connecting lines between modules, however, have no influence on the communication between modules (which is always determined via the name-based binding mechanism described earlier). Connecting lines between modules are interpreted as "comment lines" and can be used for illustration. The color of comment lines is set in the ASCET option window (see ""Colors" node" on page 55)

All the tasks required for working with CT blocks are created automatically. No additional tasks can be created for the activation of CT blocks. The `initialize` and `terminate_CT` tasks are created only once for the whole hybrid project. Additionally, a `simulate` task and an `event` task are created for each CT block in the model. These tasks are set up like any other task (see page 395).

To define the scheduling of continuous time blocks:

- Click on the "OS" tab.
- Select a CT task in the "Task" pane.
- Adjust the settings for trigger mode etc. as necessary.

You can now experiment with the hybrid project in the same way as with a project containing only modules. In the experimentation environment a solver has to be assigned to each continuous time block. Different solvers can be assigned to blocks within the same project, which means that multi-rate experiments are possible. Assigning solvers is described in section "Experimenting with Continuous Time Blocks" on page 337.

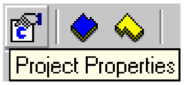
4.8.8 Project Settings

ASCET Experimentation is available for different target platforms, e.g., PC, experimental targets and microcontroller targets. For each target, a number of build, experiment code or production code options are available, as well as optimization options and ASAM-MCD-2MC generation options.

Note

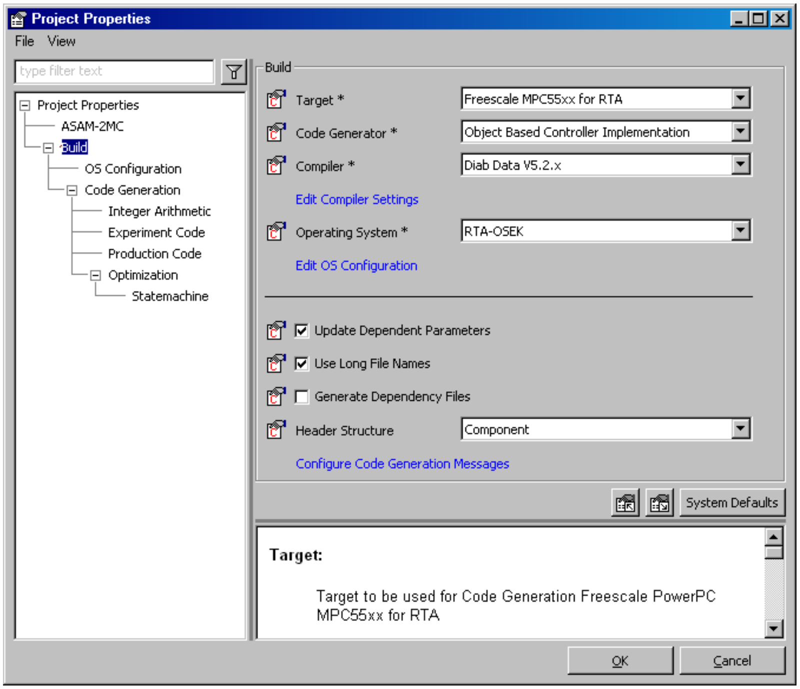
For an ASCET module, code can be generated and simulated without project context only with the `Physical Experiment` *code generator (see page 412). For the other code generators the module must be integrated into a project. A so-called default project can be defined for each class or module for that purpose. This is the only way to access the implementation information. Without project context, the conversion formulas as well as all implementations of imported entities are missing.*

To adjust the project settings:



- Click on the **Project Properties**button.

  The "Project Properties" dialog window opens in the "Build" node.



Build

Target * Freescale MPC55xx for RTA

Code Generator * Object Based Controller Implementation

Compiler * Diab Data V5.2.x

Edit Compiler Settings

Operating System * RTA-OSEK

Edit OS Configuration

[✓] Update Dependent Parameters

[✓] Use Long File Names

[ ] Generate Dependency Files

Header Structure Component

Configure Code Generation Messages

Target:

Target to be used for Code Generation Freescale PowerPC MPC55xx for RTA

- Select the desired setting in the combo box.

Or

- Activate or deactivate an option.

Or

- Enter an appropriate value in the input field.

Or

- Click on a link (blue text) to reach futhter options.
- Repeat the steps for the other options you want to adjust.

- Click on **System Defaults** to restore the default settings.

Note

The **System Defaults** *button affects* ***all*** *tabs.*

- Switch to another node to set more options,

or

- click **OK** to confirm the settings and close the dialog window.

Changes in some of the nodes below the "Build" node invalidate existing generated code in the database. The selected code generator (page 412) determines the affected nodes; the table shows the various combinations.

Code Generator	Node

`Physical Experiment`
`Quantized Physical Experiment`
	"Code Generation" (page 414)
	"Experiment Code" (page 420)
	"Optimization" (page 421)

`Implementation Experiment`	"Code Generation" "Integer Arithmetic" (page 417) "Experiment Code" "Optimization"
`Object Based Controller Implementation`	"Code Generation" "Integer Arithmetic" "Production Code" (page 421) "Optimization"

To filter project settings:

You can filter the project settings to make the display more concise.

- Filter the project settings as described in "To filter options:" on page 40.

To save and load project settings:

You can save your settings to an XML file, and load previously saved settings. However, this is possible only in the top-most node, "Project Properties".

1. Save project settings

- In the "Project Properties" window, select the "Project Properties" node.

- Click on **Export Options of Selected Node into XML file** button.

  The Windows file selection dialog opens. The output format `*.xml` is fixed.
- Enter path and name for the file.
- Click on **Save**.

  Your settings are saved to the file you specified.

2. Load project settings

- In the "Project Properties" window, select the "Project Properties" node.

- Click on the **Import Options for selected Node from XML File** button.

  A confirmation window opens.
- If you want to suppress the confirmation window in the future, deactivate the **Show next time** option (see also page 45).
- Click **OK** to confirm overwriting existing options.

  The Windows file selection dialog opens.
- Select the XML file you want to load.
- Click on **Open**.

  The settings in the specified file are loaded.

Project settings are divided into ASAM-MCD-2MC settings ("ASAM-2MC" Node) and Build options ("Build" Node). Build options belonging together are grouped into subnodes.

"ASAM-2MC" Node

The "ASAM-2MC" node offers the following settings:

- *ROM Code* toggles the generation of ROM code for experimental targets.
  When activated, the `*.cod` file contains both ROM and RAM code to enable stand-alone operation (see ASCET-RP user's guide). An executable file (`*.cod`) is generated for the experimental target.
- *Hierarchical Names* toggles between simple element names and full names in ASAM-MCD-2MC files.
  The use of full names helps to avoid name conflicts in large projects.

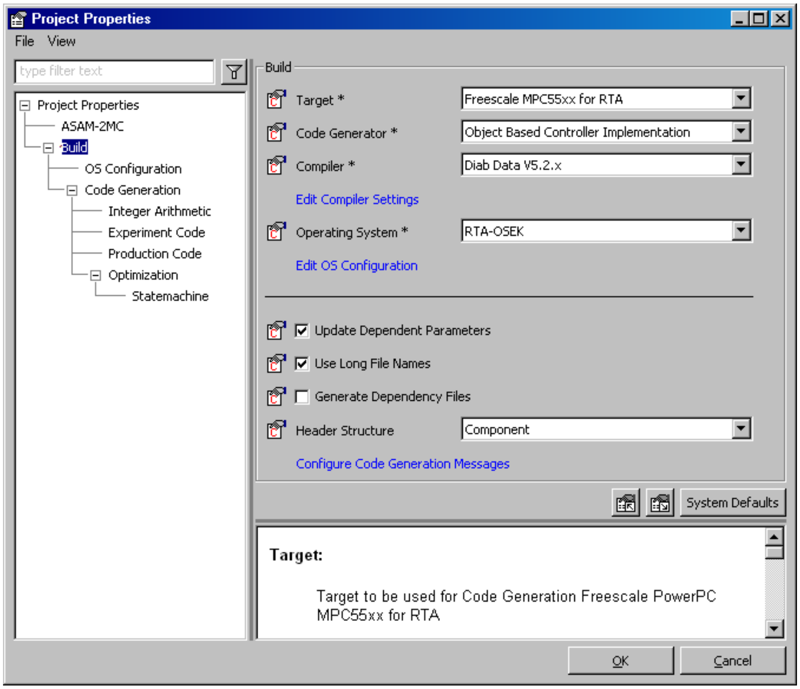
When ASAM-MCD-2MC files are generated, full names are listed in reverse order:
`<elementName>[.<className>][.<moduleName>]`

- *Formulas with Unit* defines whether the unit indicated in ASCET will be stored in ASAM-MCD-2MC together with the formulas (activated) or not (deactivated).
- *Suppress exported Parameters* determines whether exported parameters are suppressed (activated) during ASAM-MCD-2MC generation or not (deactivated).

  Some compilers optimize the access to exported parameters in the C code by replacing them with their values. When such parameters are calibrated, inconsistencies between exporting and importing components can occur. The option prohibits the calibration of such parameters.
- *Suppress exported Elements* determines whether the exported elements of prototype components are ignored (activated) during ASAM-MCD-2MC generation or not (deactivated).
- *Suppress grouping Information* determines whether the structuring of measurement and calibration variables via the `GROUP` and `SUBGROUP` keywords during ASAM-MCD-2MC generation is inserted (deactivated; default) or suppressed (activated).

"Build" Node

The following options are available in the "Build" node:



Build

Target * Freescale MPC55xx for RTA

Code Generator * Object Based Controller Implementation

Compiler * Diab Data V5.2.x

Edit Compiler Settings

Operating System * RTA-OSEK

Edit OS Configuration

☑ Update Dependent Parameters

☑ Use Long File Names

☐ Generate Dependency Files

Header Structure Component

Configure Code Generation Messages

Target:

Target to be used for Code Generation Freescale PowerPC MPC55xx for RTA

- *Target* determines the experimental or micro-controller target. The ASCET base system provides only the **PC** target, ASCET-RP adds the experimental targets, the various ASCET-SE editions add the microcontroller targets.
  The choices available here depend on the ASCET products you have installed.
- *Code Generator* determines what kind of arithmetic is used in the generated code. Available options are:
  - `Physical Experiment` (floating point arithmetic)
  - `Quantized Physical Experiment` (quantized floating point arithmetic)
  - `Implementation Experiment` (fixed point arithmetic)

  If ASCET-SE for a microcontroller target is installed on your computer, only one option is valid for that target:

  - `Object Based Controller Implementation`

- *Compiler* determines the compiler used to generate the code for the project. An appropriate compiler is supplied with ASCET for each experimental target.
- *Edit Compiler Settings* – This link opens the ASCET options window. In that window, use the appropriate subnode of the "Compiler" node to set the compiler path.
- *Operating System* shows the operating system of the current target. When you installed ASCET-SE for RTA-OSEK, and selected a respective target, you can select the RTA-OSEK version in the the edit field.
- *Edit OS Configuration* – This link opens the "OS Configuration" node (cf. page 414).
- *Use Long File Names* can be deactivated in order to guarantee that your system corresponds to the 8.3 filename pattern (filename.ext). Otherwise long filenames (identical with the corresponding element names) will be generated.

Note

This option is available only for compilers that support long file names. It is deactivated for DIAB 4.1 and Borland compilers.

- *Generate Dependency Files* switches the generation of make files containing the dependencies of all generated header files on (activated) or off (deactivated; default). Header files included by the user via `#include` statements are not taken into account. A make file is created for each generated `*.c` file.

- *Header Structure* controls the header file generation when using **Component → File Out Generated Code**.

  The following selections are available:

`Component`	A separate header file is created for each class and each module in the project. If a module contains a class, the header information of the class is inserted in the module header file. If a class is included in several modules of the project, it's header information is inserted in all module header files.
`Module`	A separate header file is created for each module in the project.
`Project`	A single header file is created for the entire project. This file contains all information from the modules and classes in the project. This header file must be included in all `*.c` files.

  Note

  Headers of internal and external C code classes and of the OS component are not affected by this option (exception: **use header global** *is activated for an external C code class). The latter is always written to the* `conf.c` *and* `conf.h` *files.*

- *Configure Code Generation Messages* –This link opens the "CodeGen message Configuration" window. In that window, you can configure the messages from code generation and build process. For a detailed description, see "Configuring Messages in the "CodeGen Message Configuration" Window" on page 154.

"OS Configuration" Node

The options in this node is only available if ASCET-SE for RTA-OSEK is installed, and a corresponding micro-controller target is selected in the build options (see page 412). They are described in the ASCET-SE user's guide.

"Code Generation" Node

This node contains the following options:

- *Protected Division* should be activated to make the generated code check for and intercept divisions by zero. This option entails a performance penalty, so it should be deactivated to achieve maximum code efficiency.

- *Generate Define Directives for Enum Values*: When this option is deactivated (default), the respective integer values for the enumerators are generated in the code.

  When the option is activated, the names of the enumerators (e.g., `Red` or `Green`) appear in the generated code, and additional macros of the following kind are generated:

  ```
  #define Red 0
  #define Green 1
  ```

  Thus, the symbolic names are assigned to the respective integer values.

- *Add Comment with Implementation Information for each Assignment Statement:* When this option is activated, a comment is generated in front of each assignment or return statement. This comment contains implementation information on the right-hand side of the assignment..

  Example:

  ```
  /* return with expr from doAddition:           ↵
        min=-65536, max=65534, hex=1phys+0,        ↵
        limit=(maxBitLength: true, assign: true)*/
  return ((sint32)input1 + input2);
  ```

- *Add Comment with Specification Source for each Statement:* When this option is activated, a comment is generated in front of each generated assignment. The comment contains the specification source of the assignment.

  – ESDL: line number

  – Block diagram: sequence call number

  – State machine: name of action/condition and respective state/transition

  Example (ESDL):

  ```
  /* doAddition: line #1 */
  self->cont->val = input1 + input2;
  ```

- *Add Comment with Generation Information for each Component:* When this option is activated, comments are created at the beginning of the generated code for each component. The first comment contains information regarding the component and ASCET, the second contains all build and experiment/production code options.

  Example:

  ```
  /**************************************************
   * BEGIN: Generation Information
   *------------------------------------------------
  ```

```
 * Component:................Project
 * Name:.....................“ControllerTest”
 * Implementation:...........“Impl”
 * Dataset:..................“Data”
 * Specification:............<not applicable>
 * Version:..................<empty String>
 *-----------------------------------------------
 * Generation Date:..........21.03.2006
 * Generation Time:..........16:34:44
 *-----------------------------------------------
 * ASCET-MD Version:.........V5.1.4
 * ASCET-RP Version:.........V5.4.1
 * ASCET-SE Version:.........V5.2.1 CID[514]
 *-----------------------------------------------
 * END: Generation Information
 ***********************************************/
/************************************************
 * BEGIN: Project Options “Build”/“Code”
 *------------------------------------------------
 *    Build
 *------------------------------------------------
 * Code Generator:.......Implementation Experiment
 * Compiler:............Borland-C V4.5
 * Operating System:....<not applicable>
 * Target:..............PC
 *------------------------------------------------
 *    Code
 *------------------------------------------------
 * add comment with generation information for   ↵
    each component [true]:.true
 ...
 *------------------------------------------------
 *    Code.Experiment
 *------------------------------------------------
 * data logging [false]:.....................false
 ...
 *------------------------------------------------
 *    Code.FixedPoint
 *------------------------------------------------
 * allow double bit size for division numerators ↵
    [true]:............true
 ...
 *------------------------------------------------
```

```
*    Code.Optimizations
*-----------------------------------------------
* common subexpression elimination [true]:....true
...
*-----------------------------------------------
* END: Project Options "Build"/"Code"
**********************************************/
```

- *Force Parenthesis for Binary Logical Operators* enforces parentheses for binary logical operators, according to the MISRA[1]-C:2004 Guidelines for the use of the C language in critical systems.
- *Add parentheses for readability* adds parentheses for better readability, according to MISRA[1] rule 12.1.

  Example: If the option is deactivated, the following code is generated:

  ```
  x = (a > b) ? a - b : a + b;
  x = a + b * c;
  if (a + b == c) {.. }
  ```

  If the option is activated, the following code is generated for the same sample operations:

  ```
  x = (a > b) ? (a - b) : (a + b);
  x = a + (b * c);
  if ((a + b) == c) {.. }
  ```

"Integer Arithmetic" Node

This node contains the following options:

- In the *Arithmetic Services Set* combo box, you select the set of arithmetic services (cf. chapter 4.14) you want to use. The combo box lists all sets available in the `services.ini` file of the current target; in addition, you can select `<None>` to switch off the use of arithmetic services.
  Use the **Edit** button to open the arithmetic services interface editor; see chapter 4.14.5.
- In the *Result on Division by Zero* combo box, you select the behavior upon division by zero. When you select `numerator` (default), the numerator is returned as result. When you select `user defined`, a

[1] **M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation, http://www.misra.org.uk

call to the user-defined C macro `protDiv0(num, result_type_max, result_type_min)` is returned. This macro is specified in one of the included header files.

Note

Make sure that the `protDiv0` *macro exists, and is correct, when you want to use the* `user defined` *option. An automatic existence or syntax check is* ***not*** *performed.*

The macro arguments have the following meaning:

– `num` is the current numerator value,

– `result_type_max` is the maximum possible value of the result data type,

– `result_type_min` is the minimum possible value of the result data type.

The second and third argument are used—in that order—to use the minimum and maximum of the result data type in the computation. It is possible, for example, to provide the macro

```
#define protDiv0(num,max,min) (num<0?min:max)
```

to get, depending on the result data type, the largest, or smallest, possible integer value.

- *Maximum bit Length (int)* determines whether the code generated for integer variables uses 8, 16 or 32 bit arithmetic in the expressions for fixed point arithmetic.
- *Maximum bit Length (float)* determines whether the code generated for floating-point variables uses 32 or 64 bit arithmetic in the expressions for fixed point arithmetic.
- *Allow Limit Service for Assignment Limitation* defines, under which conditions a limiting arithmetic service is used. If this option is deactivated, limiting services are only used if an overflow of the processor bit width has to be avoided. If the option is activated, limiting services are also used at data type boundaries with a smaller bit width, e.g. 16 bit for a 32 bit processor.
  In the first case, only overflows of the maximum possible processor type are handled by means of a limiting service. In the second case, limiting services are applied to overflows of the respective implemented data types, as well.

  For projects from earlier ASCET versions, this option is deactivated automatically during conversion.

- *Generate Limiters (may be changed locally)* creates a limit for the allocation operations if the calculated interval exceeds the interval specified for the target variable. If the option is deactivated, no limiters are generated (default: activated).
  If the option is activated, the local settings for the elements apply.
- *Allow Double bit Size for Division Numerators* allows twice the bit size set in *maximum bit length* for intermediate results of multiplications followed by a division (default: activated).
- *Use SHIFT Operation on Signed Values Instead of DIV Operation* generates right shifts instead of division operations for divisions of signed numbers (default: activated).
  Some compilers use logical instead of arithmetic right shifts. This may lead to sign errors if the option is not deactivated.
- *Use SHIFT Operation on Signed Values Instead of MUL Operation* generates left shifts instead of multiplication operations for multiplication of signed numbers (default: activated).
  Some compilers use logical instead of arithmetic left shifts. This may lead to sign errors if the option is not deactivated.
- *Generate Round Operation on float to integer Assignment* activates or deactivates rounding when a floating-point variable is assigned to an integer variable. Rounding is done by adding (positive values) or subtracting (negative values) 0.5. If the option is deactivated, the decimal places are truncated (default: activated).
- *Temp Vars always 32 bit (integer)* creates 32 bits for all temporary integer variables. When this option is deactivated, bit widths are generated flexible, depending on the actual requirements (default: deactivated).
- *Use power of 2 approximations of literals:* When the option is activated, the code generation searches, for multiplications and divisions with fractions, another fractionas approximation where either numerator or denominator is a power of 2.
  The optimization is applied only if an approximation is found that deviates at most 1 ‰ from the original fraction, and if the approximates literal induces no overflow.

  An example: If the option is deactivated (default), the following code is generated:

  ```
  result = input * (sint32)433 / (sint16)500;
  ```

  If the option is activated, the generated code becomes

  ```
  result = input * 28377 >> 15;
  ```

  The precision is almost identical in both cases, but the runtime-consuming division operation is replaced by an effective and fast shift.

Note

For microcontroller targets, these options are not available.

The following experiment code options are available:

- *Max Number of Loop Iterations* determines the maximum number of iterations the generated code will execute in a loop. This option is useful for avoiding infinite loops.
- *Prefix for Component Names* inserts a character string as prefix for the component names in the generated code.
  This option can be useful if you want multiple versions of a component in INTECRIO, which is not possible when all versions have identical names.
- *Data Logging* must be activated, if you want to use the data logger in transient mode. The data logger is described in section "The Data Logger" on page 597.
- *Protected Vector Indices* should be activated to make the generated code check for attempts to access an array or table with an out-of-range index. If, for instance, an array has five elements, the errors resulting from access to the sixth element of that array are intercepted. This option entails a performance penalty, so it should be deactivated to achieve maximum code efficiency.
- *Use OID for Generation of Component Names* specifies whether the component OID (activated) or the model name (deactivated) is used as component name in the generated code.
  If the option is deactivated, the user is responsible for avoiding name clashes.
- *Add Comment with Specification Info for each Element*: If this option is activated, a comment with specification information is generated for each element.

  Example:

```
/*----Local variables object structure-------*/
struct MODULE_C_CODE_IMPL_Obj {
    ASDObjectHeader objectHeader;
    /* model: name=cont, type=cont,              ↵
              kind=variable, scope=local,        ↵
              memory=volatile */
```

```
        real64_Obj *cont;
    };
```

- *Cache Loking*, *Cache Lock Code*, *Cache Lock Data*, *Used Cache Size*: Cache locking is only meaningful when you are working with ASCET-RP. The options are described in the ASCET-RP user's guide.

"*Production Code*" Node

These options are only available when ASCET-SE is installed, and a respective target is selected in the build options (cf. page 412). They are described in the ASCET-SE user's guide.

"*Optimization*" Node

This node contains the following options:

- *Constant Folding on Arithmetic Operations* replaces arithmetic operations with constant results with the result. As an example, the addition of two constants, `C1 + C2`, is replaced by the constant `C3` (`C3:= C1+C2`); the comparison of two constants is replaced by the result, e.g., `C1 >= C2` is replaced by `TRUE` (if `C1>=C2`) or `FALSE` (if `C1<C2`).
- *Constant Folding on Logical Operations* replaces logical operations with fixed results by the result. As an example, `X_log AND TRUE` is replaced by `X_log`, `FALSE OR X_log` is replaced by `X_log`, and `FALSE != FALSE` is replaced by `FALSE`.
- *Constant Folding on Control Flow* decides whether an expression for which the decision that the expression is always `true` or `false` can be made directly at the `If` statement, is replaced (activated; default) in the generated code or not (deactivated).
- *Operator Simplification with Respect to Intervals* simplifies expressions whose results are fixed due to interval calculations. If, for example, `X` has the interval `[0,100]`, and `Y` has the interval `[120,150]`, the expression `X < Y` becomes `TRUE` because `Y` is always larger than `X`.
- *Operator Simplification (general)* simplifies operations. For example, `X - NEG(Y)` is replaced by `X + Y`, and `NOT(NOT(X_log))` by `X_log`.
- *Common Subexpression Elimination* replaces operations with identical parts that can be simplified. For example, `X / X` is replaced by `1` , `X < X` is replaced by `FALSE`, and `X_log AND X_log` is replaced by `X_log`.

- *Optimize Direct Access Methods (one level)* should be activated to have direct access methods resolved in the generated code. Otherwise, a function call is generated for each direct access method call. The option should be deactivated if problems occur during code generation.

Note

For classes specified as method arguments, ***optimize direct access*** *to their elements is not supported.*

- *Optimize Direct Access Methods (multiple levels)* should be activated to have nested direct access methods resolved in the generated code. Nesting direct access methods is possible only when specifying components in ESDL. The effect of this option is the same as for the optimization of simple method calls.

"*Statemachine*" *Node*

This node contains the following options:

- *Outline Generated Methods(may be changed locally)*: If identical code is required for different transitions out of the same hierarchy state, a separate method is created (Outlining) for this code under certain conditions. This method is called wherever required.

  If you do not want to create separate methods and method calls, deactivate the option. In that case, the action code is inserted directly wherever required.
- *Auto-inline private methods (smaller code-size - may be changed locally):* If small private functions (or functions with a low number of callers) are inlined, both code size and runtime are saved.
  If the option is activated, the code generator recognizes such functions automatically and directs the compiler to inline them.
- *Hierarchical Code-Generation (may be changed locally):* Code for state machines can be generated either flat (one `switch` statement with a `case` expression for each base state) or hierarchical (nested `switch` statements according to the state hierarchy). Flat code generation (deactivated) optimizes for runtime, hierarchical code generation (activated) optimizes for code size.

- *Optimize Static Actions (Restricted Modeling)*: When the option is deactivated (default), code for the static action of a hierarchy state is generated separately for each transition that does not leave the hierarchy state.

  When the option is activated, the static action of a hierarchy state is generated only once for each substate. Thus, the code size is reduced.

Note

Optimization with this option changes the order for the execution of actions and evaluation of conditions—and thus possibly the state machine behavior. In addition, this optimization is not possible for some models.
A more detailed description is given in chapter "Optimizing the State Machine" in the ASCET reference guide.

- *Generate well-formed switch:* If the option is activated, state machine code is generated that complies with the MISRA[1] rules 14.7, 15.2, and 15.3.
- *Initialize history variable with zero*: If the option is activated, the history variables of hierarchy states in the project are initialized with `0` instead of the respective start state.

  This causes an additional assignment in the code and may violate the MISRA rule 15.3.

4.8.9 Defining the Implementation for Fixed Point Arithmetic

When generating production code for microcontrollers, the arithmetic of the physical ASCET specification often has to be mapped to fixed point arithmetic, since many microcontrollers used in electronic control units do not support floating point arithmetic. This mapping is described by the implementation transformation, or implementation for short.

Like data sets, each component or project may have any number of implementations. Like data sets, implementations can be viewed, edited, renamed, deleted, browsed, exported (both flat and recursive), added, and copied.

As with data sets there are special editors for the basic elements which form the leaves of the hierarchical tree structure of a project or component. Implementation of the basic elements of types `continuous` or `discrete` requires specification of a range for the physical values and a transformation formula.

1. **M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation, http://www.misra.org.uk

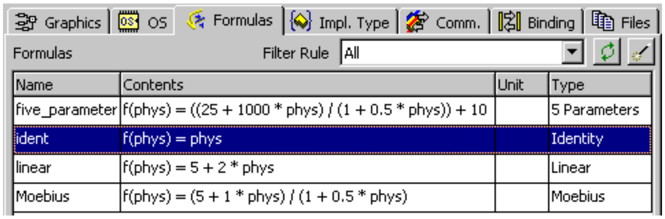
Many elements within a project usually have the same transformation formula, for instance all the velocities processed in the embedded software. It is possible to define formulas only once within a project, and then use them across all the elements in that project. Formulas can also be exchanged between projects with an import/export mechanism.

ASCET also supports the intermediate step of generating quantized floating point code. With this type of code fixed point arithmetic is emulated, but the links and the quantization of the fixed point arithmetic can be changed interactively during execution of the code. This intermediate step allows development of the optimum implementation for the elements in a specific project.

Formulas

To add a formula:

- In the project editor, click on the "Formulas" tab.



Name	Contents	Unit	Type
five_parameter	f(phys) = ((25 + 1000 * phys) / (1 + 0.5 * phys)) + 10		5 Parameters
ident	f(phys) = phys		Identity
linear	f(phys) = 5 + 2 * phys		Linear
Moebius	f(phys) = (5 + 1 * phys) / (1 + 0.5 * phys)		Moebius

The `ident` formula is always there. It is selected by default in the implementations of newly created elements.

- Select **Global Formulas** → **Add** to create a new formula.
- Type in a name for the formula and press <ENTER>.

To edit the formula:

- In the "Formulas" tab, select the formula you want to edit.
- Select **Global Formulas** → **Edit**

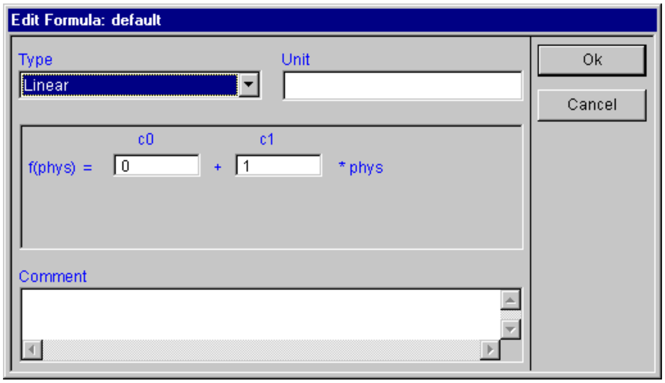
or

- select **Edit** from the context menu

or

- double-click on the formula.

  The formula editor for global formulas opens..



- In the "Type" combo box, select the type of formula to be edited.
- Enter the required values in the respective fields.
- Enter a unit into the "Unit" field.

  The unit is used for documentation purposes only and does not influence the functionality.
- Type a comment into the "Comment" field.
- Click **OK**.

You should not rename or delete the `ident` formula. When you edit the ident formula and select another type from the "Type" combo box, the following warning is displayed:

```
The <ident> formula is used for newly created elements
to provide an identity conversion and changing this
formula may corrupt your implementation. Do you
really want to change the <ident> formula?
```

When you really want to change the type of the `ident` formula, click **Yes**. Otherwise, click **No**.

The transformation formula for the implementation must be a linear formula. Four types of conversion formulas are supported for displaying this formula. The display of the formula editor adjusts to reflect the formula type you choose.

- The *Identity* formula represents an identity mapping.

$$f(phys) = phys$$

- The *Linear* formula represents a linear mapping with the coefficients `c0` (offset) and `c1` (gradient).

$$f(phys) = \overset{c0}{\boxed{0}} + \overset{c1}{\boxed{1}} * phys$$

- The *Moebius* formula represents a rational mapping as the quotient of two linear mappings, with the coefficients `c0` (numerator offset), `c1` (numerator gradient), `d0` (denominator offset), and `d1` (denominator gradient). The Moebius formula is mainly used to avoid a conversion to the linear formula format when the specification of a linear formula is given in the Moebius formula format.

$$f(phys) = \frac{\overset{c0}{\boxed{0}} + \overset{c1}{\boxed{1}} * phys}{\underset{d0}{\boxed{1}} + \underset{d1}{\boxed{0}} * phys}$$

- The *Five Parameters* formula also represents a rational mapping of the same kind in a different format with the coefficients `p1`, `p2`, `p3`, `p4` and `p56`. This formula is also used to avoid a conversion to the linear formula format when the specification of a linear formula is given in the five parameters formula format.

$$f(phys) = \frac{\overset{p2}{\boxed{0}} + \overset{p1}{\boxed{1}} * phys}{\underset{p4}{\boxed{1}} + \underset{p3}{\boxed{0}} * phys} + \overset{p56}{\boxed{0}}$$

To rename a formula:

- In the "Formulas" tab, select a formula.
- Select **Global Formulas** → **Rename**

or

- select **Rename** from the context menu to rename the formula.

To delete a formula:

- In the "Formulas" tab, select one or more formulas.
- Select **Global Formulas** → **Delete**

or

- select **Delete** from the context menu to delete the formula.
- In the "Confirm" window, confirm the command with **OK**.

  The formulas are deleted. A list of the deleted formulas is shown in the ASCET monitor window.

To sort the display:

- Click on a column name.

  The display is sorted by that column. The columns are sorted alphabetically, except "Type". That column is sorted as follows:

  Identity → Linear → Moebius → 5 Parameters

  A second sorting reverts the sorting order.

To filter the display:

You can apply several filters to the formulas displayed in the "Formulas" tab.

- In the "Filter Rule" combo box, select a filter.

- The following filters are available:

Flter	displayed formulas
`All`	all formulas
`Used in ACTIVE implementations`	formulas used to implement an element in the project
`Not used in ACTIVE implementations`	formulas *not* used to implement an element in the project
`Used in ANY implementation`	formulas either used to implement an element or used within a currently unused implementation
`Not used`	unused formulas

  If you select, e.g., the `Not used` filter, you can easily delete unused formulas.

- Click on **Update** to update the formula display.

  This is necessary when you have edited the implementation of the project or one of its components in another editor.

To add missing formulas:

If elements in the project use formulas that are not defined in the project, code generation produces error messages. To add the missing formulas, proceed as follows.

- In the project editor, "Formulas" tab, select **Global Formulas** → **Add missing**

or

- click on **Add missing formulas**.
- For each undefined formula in the active implementation, a new formula with the respective name, and behaving like the identity, is added.

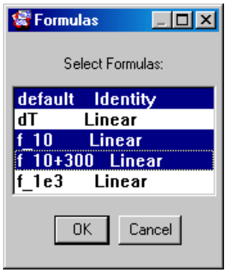
To import formulas:

- In the project editor, click on the "Formulas" tab.

- Select **Global Formulas** → **File In** → **Select** or **All**.

  The Windows file dialog box opens.
- Select the file that contains the formulas you want to add.
- Click on **Open**.

If you have selected the **All** command, all formulas in the selected file are imported. Otherwise you can choose the formulas to import from a dialog box.



default	Identity
dT	Linear
f_10	Linear
f_10+300	Linear
f_1e3	Linear

To file out formulas:

- Select **Global Formulas** → **File Out** → **Selected** or **All**.

  The Windows file dialog box opens.
- Select a directory and filename.
- Click **Save**.

If you have selected the **All** command, all the formulas in the current project are filed out. Otherwise only the formula selected is exported. Formulas are filed out in ASCII format, which can be read with any text editor.

An array has one implementation which is defined like that of a scalar element. A 1-D table has two implementations, one for the input and one for the output value. A 2-D table has three implementations, two for the inputs and one for the output value.

Global Changes in Implementations

When a formula is edited or an existing formula has been replaced by another one, you can now update your implementations recursively. Also, you can replace a formula throughout all the implementations of a project to adjust your implementation settings.

Changes are not propagated automatically, you must update your implementations whenever a formula has either been replaced or modified.

To replace a formula recursively:

- In the project editor, activate the "Formulas" tab so you can see a list of existing formulas.
- Select **Extras** → **Global Replace Formula**.

  A dialog box prompts for the name of the old and the new formula.
- Enter the names as needed and click **OK**.

  The formula is replaced recursively in the entire implementation.

To update all implementations:

- In the project editor, select **Extras** → **Update Implementations** to make changes to the formulas in all the implementations.

  A protocol of changes is written to the ASCET monitor window. Changes in the implementations are made according to the master page selected in the implementation editor of each primitive element.

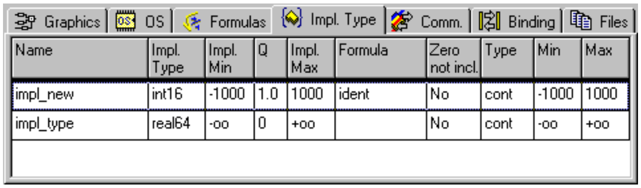
Implementation Types

To be able to edit the implementations of individual variables more easily and to be able to easily assign the same implementations to elements with comparable physical significance, you can define what are referred to as *implementation types* in the project context. This is also true of the default project (see chapter 4.8.1) of a class or a module. These implementation types contain the essential specifications of an implementation and can be assigned to individual elements in their implementation editors.

This section describes how to create and set up implementation types. How these are used during implementation is described in the section "Using Implementation Types" on page 491.

To add an implementation type:

- In the project editor, click the "Impl. Type" tab.



Graphics | OS | Formulas | Impl. Type | Comm. | Binding | Files

Name	Impl. Type	Impl. Min	Q	Impl. Max	Formula	Zero not incl.	Type	Min	Max
impl_new	int16	-1000	1.0	1000	ident	No	cont	-1000	1000
impl_type	real64	-oo	0	+oo		No	cont	-oo	+oo

- Select **Global Implementations** → **Add** → ***<model type>***

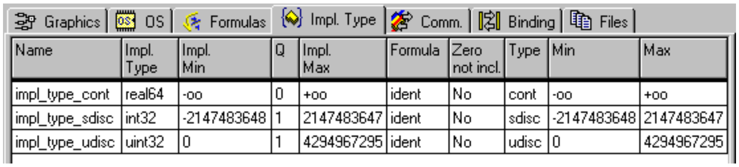
or

- select **Add** → ***<model type>*** from the context menu.

Note

The model type you select when generating an implementation type, determines which variables you can assign this implementation type (see "Using Implementation Types" on page 491).

A new implementation type is created with a default name. Automatic settings (see following figure) are selected which correspond to your selection for ***<model type>***.



Graphics | OS | Formulas | Impl. Type | Comm. | Binding | Files

Name	Impl. Type	Impl. Min	Q	Impl. Max	Formula	Zero not incl.	Type	Min	Max
impl_type_cont	real64	-oo	0	+oo	ident	No	cont	-oo	+oo
impl_type_sdisc	int32	-2147483648	1	2147483647	ident	No	sdisc	-2147483648	2147483647
impl_type_udisc	uint32	0	1	4294967295	ident	No	udisc	0	4294967295

To manage implementation types:

You can delete several implementation types simultaneously, but you have to rename them one by one.

- Select an implementation type.

- Select **Global Implementations** → **Rename**

or

- select **Rename** from the context menu.

  The corresponding cell in the "Name" column becomes an input box.
- Enter a name for the implementation type and press <RETURN>.
- Select **Global Implementations** → **Delete**

or

- select **Delete** from the context menu.

  The selected implementation type is deleted.

To edit implementation types:

- Select the implementation type you want to edit in the "Impl. Type" tab.
- Select **Global Implementations** → **Edit**

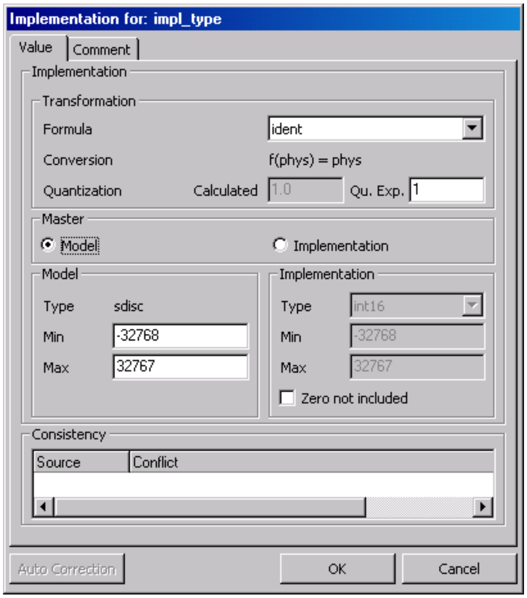
or

- select **Edit** from the context menu

or

- double-click the implementation type.

  The implementation editor for implementation types opens in the "Value" tab.


Implementation for: impl_type

Value | Comment

Implementation

Transformation

Formula: ident

Conversion: f(phys) = phys

Quantization — Calculated: 1.0 — Qu. Exp.: 1

Master: (● Model) ○ Implementation

Model — Type: sdisc; Min: -32768; Max: 32767

Implementation — Type: int16; Min: -32768; Max: 32767; ☐ Zero not included

Consistency

Source	Conflict

Auto Correction | OK | Cancel

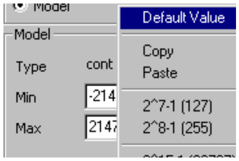
- Select a conversion formula in the "Formula" dropdown list.
- If necessary, enter the quantization in the "Qu.Exp" field.

  The field is only activated if you selected the `Quantized Physical Experiment` code generator in the build options (see page 412). In all other cases, it is only used as a display.
- Select the master page of the implementation in the "Master" field.

1. Master **Model**:

   Select this setting if physical model properties are the main reason for selecting the implementation.
   All fields in the "Model" area except model type can be edited. This is

determined when the implementation type is created.
The "Implementation" area is locked apart from the option **Zero not included**.

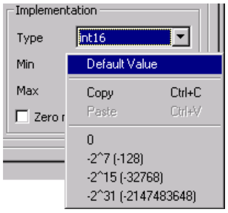


- Enter the limits for the interval in the "Min." and "Max." boxes.
- To set the upper or lower bound of a model type ("infinite" in case of "cont"), click one of the fields with the right-hand mouse button and select **Default Value** from the context menu.

  The values for the implementation are updated automatically.

2. Master **Implementation**:

   Select this setting if certain code properties (e.g. a required bit width) are the main reason for selecting the implementation.
   The fields in the "Implementation" area can be edited; the "Model" area is locked.



- Select the type of implementation in the "Type" dropdown list.
- Enter the limits for the interval in the "Min." and "Max." fields.
- To set the default limits for the current data type, click one of the fields with the right-hand mouse button and select **Default Value** from the context menu.

  The values for the model are updated automatically.

3. Other settings:

- If the interval *cannot* have the value "zero", activate the option **Zero not included**.

Note

If you activate the option **Zero not included***, no check is carried out in the generated code during runtime to see if the denominator of a division is zero. This can lead to critical run-time behavior of the code. If in doubt, disable the option.*

- Select the "Comment" tab if you want to enter a comment.

- Enter the comment in the text box of the tab.

  This comment is *not* transferred during assignment to an element.
- Click **OK** to confirm your settings.

You cannot make any settings for the memory area and limiting behavior for an implementation type during assignments or overflow or enter any additional information. These settings are made individually for each element.

To copy implementation type settings:

Once you have set up an implementation type, you can copy the settings to another implementation type.

- Select the implementation type whose settings you want to copy.
- Select **Global Implementations** → **Copy**

or

- select **Copy** from the context menu.

  The settings are copied to the database clipboard.
- Select the implementation type into which you want to copy the settings.
- Select **Global Implementations** → **Paste**

or

- select **Paste** from the context menu.

  The settings are copied from the database clipboard into the implementation type.

To swap settings of implementation types and scalar elements:

You can also swap the settings between implementation types and scalar elements of the database.

- In the "Impl. Type" tab of the project editor, select the implementation type whose settings you want to copy.
- Select **Global Implementations** → **Copy**

or

- select **Copy** from the context menu.

  The settings are copied to the clipboard.

1. Via the "Implementation" tab of the Component Manager

   - Select the component which contains the scalar element to which you want to copy the settings from the "1 Database" list in the Component Manager.
   - Select the "Implementation" tab from the "3 Contents" field.
   - Select the element from the "Implementation" tab.
   - Select **Paste** from the context menu

   or

   - press <CTRL> + <V>.

2. Via the implementation editor of a component

   - Open the implementation editor of the component that contains the scalar element to which you want to copy the settings.
   - Select the tab which contains the element.
   - Select the element.
   - Select **Element** → **Paste Implementation From Buffer**

   or

   - select **Paste Implementation From Buffer** from the context menu.

     The settings are copied from the clipboard to the scalar element.

If you want to copy the implementation of a scalar element into an implementation type, proceed in the reverse order: copy the implementation in the Component Manager or in the implementation editor and add it to the "Impl. Type" tab of the project editor.

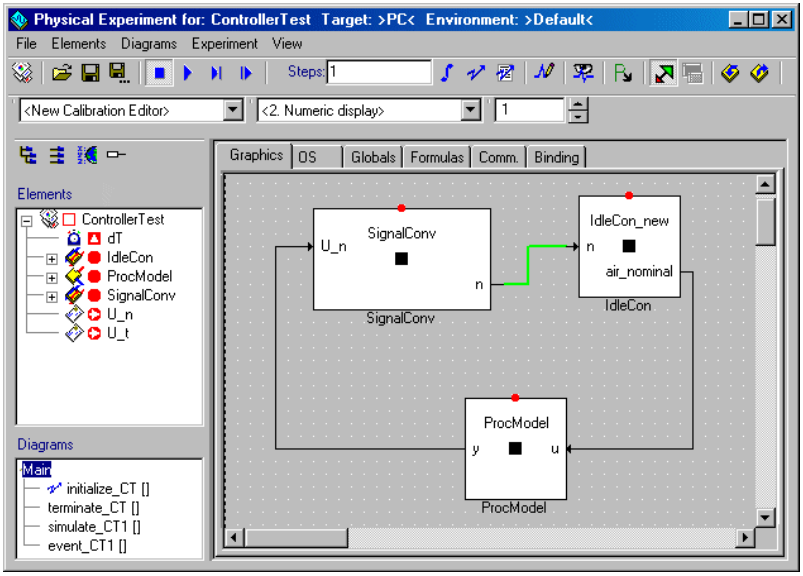
4.8.10 Experimenting with Projects

The code generation and experimentation facilities are much more powerful for projects than for components. Only offline experimentation with floating point arithmetic (build option `Physical Experiment`, see page 412) is available for components. Projects offer online experimentation, and both offline and online experimentation can be combined with either floating point arithmetic or quantized floating point arithmetic (build option `Quantized`

`Physical Experiment`) or fixed point arithmetic (build option `Implementation Experiment`). The same code generation facilities are available for components, when they are experimented with in a project context.

Online and Offline Experimentation

It is possible to experiment with a project offline just as with any component. All facilities for offline experimentation are available and work in the same way as for components. Offline experimentation with a component is equivalent to project offline experimentation, with the default project containing the component. The offline experimentation environment is described in detail in chapter "The Experimentation Environment" on page 561.



The main difference between offline experimentation with components and projects is that in projects, the tasks created in the operating system editor are stimulated in the event generator, rather than the methods or processes of any components.

With online experimentation a project can be tested under more realistic conditions and in real-time. 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. During online experimentation the code always runs on an experimental target in real-time. Online experimentation forms the basis for using ASCET in practical applications. It focuses on the operating system schedule and the corresponding real-time behavior of the control system, whereas offline experimentation often revolves around testing the functional specification of a system.

The main differences between online experimentation and offline experimentation are:

- Online experiments have to be run on real-time hardware, e.g. the ETAS targets ES1130 and ES1135.
- There is no stimulation, i.e. there are no event and data generators. The scheduling is determined by the integrated operating system and the data is read from e.g. the technical process.
- The sampling rate at which data is measured must be specified explicitly, by selecting an activation task for each measurement window.
- The experiment and the measurements can be started independently of each other.

Online experiments are possible only in connection with ASCET-RP; therefore, you find the description in the ASCET-RP user's guide.

It is possible to generate code for the project without starting the experimentation environment. This is useful for testing whether the code generated is syntactically correct. If the code is incorrect, the program displays the error messages issued by the compiler.

To generate code:

- In the project editor, select **Component** → **Generate Code**

or

- click on **Generate Code**.

  The code for the project is generated. Any error messages are displayed in a message window.

Or

- Select **Component** → **View Generated Code**.

  C code is generated and displayed automatically in a text editor.
  The editor can be selected in the ASCET Options window, "ASCII Editor node" (cf. page 60).

To save generated code:

- Select **Component** → **File Out Generated Code** → **Flat** or **Recursive** to write the code for a project to a file.

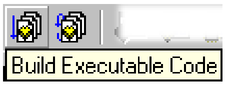
  The Windows path selection window opens.
- Select a path.
- Click **OK** to write the code to the directory you selected.

  The generated code is written to the specified location. You can open the file with any text or C code editor. The names of the generated files are logged in the monitor window.

To generate executable code:

- In the project editor, select **Component** → **Build**

or



- click on **Build Executable Code** to generate an executable file.

  The code for the entire project is gebnerated, compiled, ans linked. When no errors occur, an executable file is generated.
  The source and object code generated in that process is stored in the ASCET database.

Or

- Select **Component** → **Build from directory** to generate an executable file without storing the generated code in the ASCET database.

  The time required for storing the files is thus saved.

  This option can be necessary particularly when generating code for large projects, in case the computer does not have enough main memory.

When an executable file is generated, all files (including source code) are written by default into the `.\cgen\` directory.

Note

When the **Keep files in Code Generation Directory** *option is deactivated in the options window (cf. page 49), the content of the ASCET code generation directory* `.\cgen\` *is deleted whenever you exit your ASCET session. To store them permanently, you have to file them out as described on page 439.*
Retrospectively activating the option has no effect for the running session.

If you want, you can link existing object files to an executable file without new code generation and compilation. This is primarily of interest when your project contains external C code files which are integrated during the make process.

To link object files to an executable file:

- Perform the necessary changes in the external files, and create object files.
- In the project editor, select **Component** → **Link Only**.

  The internal object files belonging to the project are written from the database to the hard disk and, together with the external object files, linked to an executable hex file.

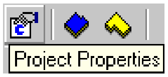
Note

Link Only *works corrdctly only when the internal files have been created using* **Component** → **Build** *or the* **Build Executable File** *button. Only with these commands, the files are stored* ***in the database***.

Experimenting with Quantized Floating Point Code

Code with quantized floating point arithmetic is generated from the same physical model as code with floating point arithmetic. You only need to switch the code generation over.

To switch to quantized floating point arithmetic:



- Click on the **Project Properties**button.
- In the "Code Generator" option select `Quantized Physical Experiment`.

- Click **OK**.

  The next time you start the experiment, your code is generated with the current settings.

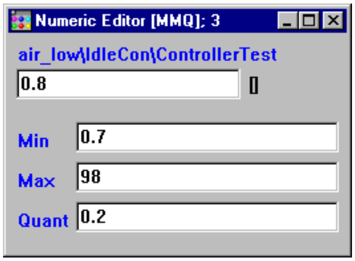
Experimenting with models with quantized floating point arithmetic works in the same way as with standard floating point arithmetic. The difference is that for experimentation with quantized floating point arithmetic, there is a special calibration window to adjust the limits of the interval and the quantization of the value.

To open the quantized calibration window:

- In the "Elements" pane of the experimentation environment, select the element you want to calibrate.
- Select **Elements** → **Calibrate**.

  A dialog box prompts for the type of editor you want to use.
- From the list, select the `[Numeric Editor [MMQ]]` and click **OK**.

  The quantized calibration window opens with the values of the current data set and the values from the current implementation.



0.8

Min 0.7

Max 98

Quant 0.2

Quantized floating point arithmetic only affects the assignment to variables. Each time a variable is written or its value is changed in the calibration window, the value is adjusted to the quantization information.

To work with the quantized calibration window:

- Select the value field.
- Change the value as you wish.

- Press <ENTER>.

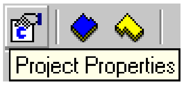
  The value is changed in the experiment and automatically adjusted to the quantization and interval chosen.

- Select one of the fields "Min", "Max" or "Quant" and change the value to what you want.
- Press <ENTER> or select another field.

  The new information is transferred to the experiment and the value is adjusted automatically to the new interval and the quantization.

Quantized floating point arithmetic can also be used for experimentation with components, but only with components not described in C code. The component to be experimented with must be opened from the project editor, instead of from the database browser.

To experiment with a component in quantized floating point arithmetic:



- In the project editor click on the **Specify Code Generation Settings** button.
- In the "Code Generator" option select `Quantized Physical Experiment`.
- Select the component in the "Elements" pane.
- Select **Element** → **Edit Component**.
- The relevant specification editor opens.
- Experiment with the component.

4.8.11 Generating Application Data

In the project editor, you can generate all files required for running your code on a controller target, and measuring/calibrating it with a calibration system (e.g., INCA). ASCET uses the ASAM-MCD norm to generate information needed in the calibration system. The ASAM-MCD file represents the interface to all calibration systems that recognize the standard ASAM-MCD format.

To generate application files:

- In the project editor, select **ASAM-2MC** → **Write** to create the ASAM-MCD-2MC files for your system.

  A file prompter dialog box is displayed.
- Select a path and enter a filename for the ASAM-MCD-2MC files.
- Click **Save** to have the application files written to the file system.

  ASCET generates the ASAM-MCD-2MC data file and the hexadecimal file that contains the code for your system.

You can set options for the ASAM-MCD-2MC file generation in the "ASAM-2MC" node of the "Project Properties" window. The procedure is described in section "Project Settings" on page 407.

4.9 Containers

The Containers available since ASCET 5.0 are used 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. When upgrading from an earlier ASCET version, networks are converted to containers (see chapter 4.9.2). Unlike the networks in earlier ASCET versions, containers have no other function.

Containers can contain all kinds of database items apart from folders, even other containers. Direct (the container contains itself) and indirect (the container contains a second container, which in turn contains the first) recursions are admissible. Each database item can only be contained once in the same container.

4.9.1 Working with Containers

Containers are (like enumerations) not edited in a special editor but in the Component Manager. A context menu, similar to those of other views, is available to you there in the container view in the "Container components" tab.

To invoke the container view:

- In the Component Manager, mark an existing container in the "1 Database" field

or

- create a new container (see page 75).

  The "Container components" tab is displayed in the "3 Contents" field.
- Click anywhere in the "Container components" tab to bring it into focus.

To add a database item (context menu/keyboard):

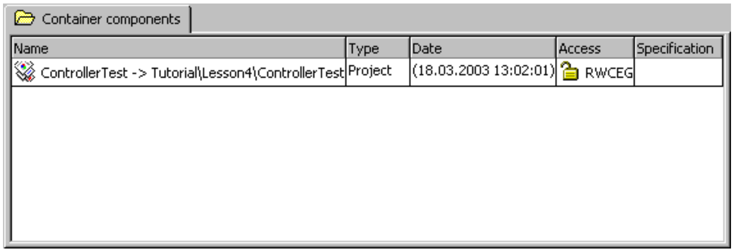
- Select **Add** from the context menu

or

- press <INS>.

  The "Select Item" window opens.
- Select the component you want to add from the "1 Database" list.
- Click **OK** to add the item and close the "Selected Items" window.

  The item is added to the container and displayed in the "Container components" tab.



Container components

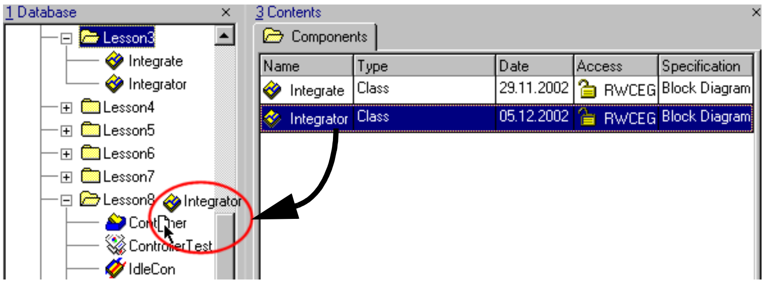
Name	Type	Date	Access	Specification
ControllerTest -> Tutorial\Lesson4\ControllerTest	Project	(18.03.2003 13:02:01)	RWCEG	

To add a database item (Drag & Drop):

- In the "1 Database" list of the Component Manager, expand the folder containing the container to which the item is to be added.
- In the "1 Database" list, select the folder containing the database items to be added to the container.

  The content of the folder is displayed in the "3 Contents" field.

- Drag the database items from the "Components" tab to the container in the "1 Database" list.

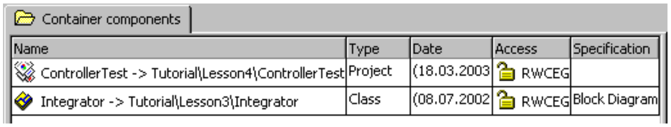


Name	Type	Date	Access	Specification
Integrate	Class	29.11.2002	RWCEG	Block Diagram
Integrator	Class	05.12.2002	RWCEG	Block Diagram

Note

*You **cannot** drag the database item from the "1 Database" list into the container.*

The items are added to the container.



Name	Type	Date	Access	Specification
ControllerTest -> Tutorial\Lesson4\ControllerTest	Project	(18.03.2003	RWCEG	
Integrator -> Tutorial\Lesson3\Integrator	Class	(08.07.2002	RWCEG	Block Diagram

You can drag items from one container to another in the same way. The items are not removed from the first container.

To rename added items:

Note

If you rename an item in a container, the original database item is also renamed; this is not the case when items added to components or projects are renamed.

- Mark the item you want to rename in the "Container components" tab.
- Select **Rename** from the context menu

or

- press <F2>.

ControllerTest | Project

The name of the item is highlighted.

- Enter a new name and press <RETURN>.

  The item is renamed both in the container and in the database.

To edit items:

You can open an item you want to edit from within the container the same way as from a folder.

- Mark the item you want to edit.
- Select **Component** → **Edit Item**

or

- select **Edit** from the context menu

Note

These two options are not available for containers contained in containers.

or

- press <RETURN>.

or

- double-click the item to open the component editor.

  If the item is a container, it is selected in the "1 Database" field and displayed in the "3 Container components" field.

To delete items from the container:

- Mark the items you want to delete in the "Container components" tab.
- Select **Select All** from the context menu

or

- press <CTRL> + <A> if you want to select all items.
- Select **Delete** from the context menu

or

- press <DEL> to delete the selected items from the container.

  The items are only deleted from the container, not from the database.

To sort the container view:

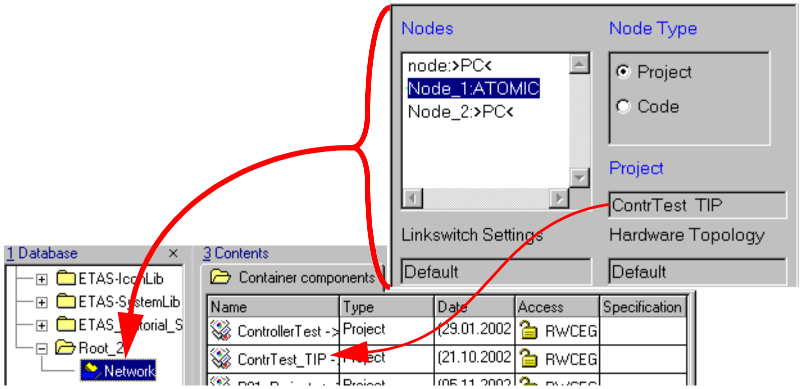
- Select **Sort by** → ***<column>*** from the context menu

or

- click on the name of a column to sort the display according to the relevant column.

4.9.2 Containers and Networks

The containers in ASCET 5.2 replace the networks of earlier ASCET versions. If you open an old database which contains networks, these networks are automatically converted into containers. The projects assigned to the different nodes of the network are added to the containers. Other information available in the network is *not* added.



Nodes: node:>PC<, Node_1:ATOMIC, Node_2:>PC<

Node Type: [✓ Project] Code

Project: ContrTest_TIP

Linkswitch Settings: Default

Hardware Topology: Default

1 Database: ETAS-IconLib, ETAS-SystemLib, ETAS_Tutorial_S, Root_2, Network

3 Contents: Container components

Name	Type	Date	Access	Specification
ControllerTest ->	Project	29.01.2002	RWCEG	
ContrTest_TIP ->	Project	21.10.2002	RWCEG	

4.10 Editing Element Properties

Elements are interface elements, variables, tables, arrays and complex elements (referenced components).

Each element has a number of properties, which can be configured according to requirements in the model.

The configuration selected is valid for all occurrences of an element within a component.

Instances and Occurrences

An element can have several occurrences. An occurrence of an element is the equivalent to writing down the name of the element in a text-based programming language. Changes in one occurrence of an element affect all the other occurrences of that same element.

Care has to be taken when using referenced components as elements. Here, every component can have multiple instances, each of which can in turn have multiple occurrences.

4.10.1 Element Configuration

An element configuration is edited in the element editor. The element editor can be called from any specification editor and from the Component Manager.

To open the element editor:

- In the "Elements" pane of a specification editor, highlight the element you want to edit.
  - Select **Element** → **Edit**

or

  - right-click on the element and select **Edit** from the context menu

or

  - double-click the element you want.

Or

- In the drawing area (BDE) or "Graphics" tab (project editor), right-click on the element and select **Edit** from the context menu.

Or

- In the Component Manager or the Browser view of the specification editor, select the "Elements" tab.
- Highlight the element you want to edit.
  - Press <ENTER>.

or

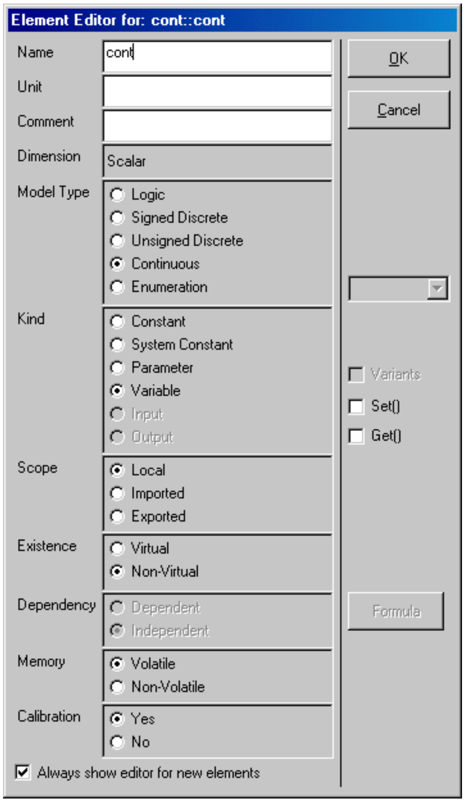
  - Right-click on the "Elements" tab and select **Edit** from the context menu.

The "Element Editor" window opens.

When the **Edit Primitive Elements** (or **Edit Implementation Cast**) option in the ASCET option dialog (cf. "Options for Confirmation Dialogs" on page 45) is activated, the element editor opens automatically when a new element is created.

This option can be overwritten with the **Always show editor for new elements** option present in each element editor.

Depending on the type of the selected element, the element editor can look differently; not all fields exist in every case. The functions of the available fields are identical in all cases.

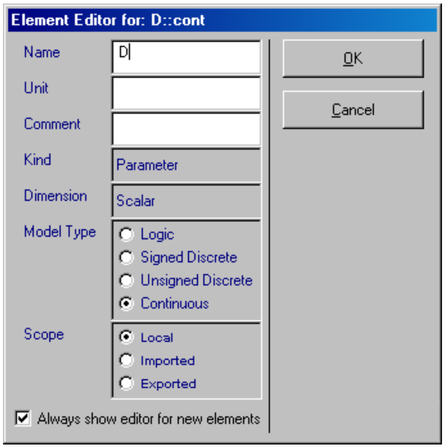


Element Editor for: cont::cont

Name	cont
Unit	
Comment	
Dimension	Scalar
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

Fig. 4-2 Element editor for basic elements. Depending on the element, some fields or options can be deactivated.


Element Editor for: D::cont

Name: D

Unit:

Comment:

Kind: Parameter

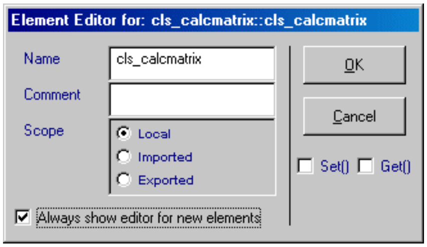
Dimension: Scalar

Model Type: Logic [✓ Continuous] Signed Discrete Unsigned Discrete

Scope: [✓ Local] Imported Exported

☑ Always show editor for new elements

Abb. 4-3 Element editor for CT block elements. The possibilities to set Get and Set ports, as well as some other fields, do not exist, other fields are used as displays only.


Element Editor for: cls_calcmatrix::cls_calcmatrix

Name: cls_calcmatrix

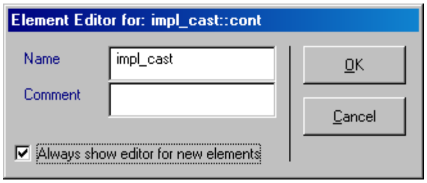
Comment:

Scope: [✓ Local] Imported Exported

☐ Set() ☐ Get()

☑ Always show editor for new elements

Abb. 4-4 Element editor for an included component. Only a few fields exist.


Element Editor for: impl_cast::cont

Name: impl_cast

Comment:

☑ Always show editor for new elements

Abb. 4-5 Element editor for an implementation cast. Only the name and the comment can be edited.

To edit an element configuration:

Configure the element as you wish. Proceed as follows:

- In the element editor, enter a unit into the "Unit" field.

  Each element can be assigned a unit. The unit is for documentation purposes only, and does not influence the functionality of the element.

- Type a comment into the "Comment" field.

  The comment is displayed when documentation is generated for the component.

- Assign the element type in the "Model Type" field.

  When you select the **Enumeration** type, the combo box next to the "Model Type" field (see Fig. 4-2) is activated. It contains all enumerations available in the database.

- In the "Kind" field, assign the element kind.

  Here, you can define constants or system constants (cf. "The Kind of Elements" in the ASCET reference guide).

  The options **Input** and **Output** are only available for inputs and outputs of state machines.

- In the "Scope" field, determine the scope of the element (cf. "The Scope of Elements" in the ASCET reference guide).

- In the "Existence" field, determine whether the element is virtual or not.

  This field is only available for variables and parameters (enumerations excluded).

- In the "Dependency" field, determine whether a parameter is dependent or independent.

  This field is only available for parameters (enumerations excluded).

- In the "Memory" field, determine whether the element is written to the **volatile** or **non-volatile** memory of the ECU.

  Data stored in the non-volatile memory of the ECU will not be overwritten upon initialization.

Note

*The `sm` state variable of a state machine cannot be edited in the element editor. To assign the **non-volatile** attribute to this variable, open its context menu in the "Elements" pane and select* **Settings** → **Non-Volatile**.

- In the "Calibration" field, determine whether a parameter can be calibrated in an application system (**Yes**) or not (**No**).

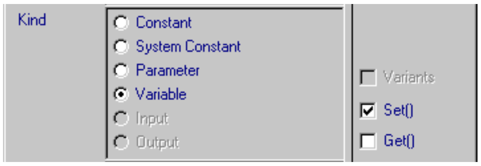
  When you select **No**, the `READ ONLY` keyword is generated for the parameter during ASAM-MCD-2MC generation.

  This field has *no* influence on other elements or the offline experimentation environment.
- Click **OK** to close the element editor.

It is possible to enable individual elements within a component. Enabling an element means that the element can be accessed from outside of the current component.Parameters and constants can be read out from the current component. Variables, however, can be read in and out.

To enable or disable elements:

- Open the element editor.



Kind

- ( ) Constant
- ( ) System Constant
- ( ) Parameter
- (●) Variable
- ( ) Input
- ( ) Output

- [ ] Variants
- [✓] Set()
- [ ] Get()

- Click on the **Set()** option.

  A tick mark appears in the box. The element is enabled and can be written from outside the component.

- Click on the **Set()** option again to reverse the setting.
- Activate the **Get()** option to add an output.
- Click **OK**.

  In the layout, an input or output is added for the element.

You have to actively set the **Get()** *and* **Set()** *options for messages to add the respective inputs and outputs to the module layout. However, these are merely a visualization feature; assignment is performed using identical names.*

When you are using Get/Set ports without activating one of the **Optimize Direct Access Methods (...)** *code optimization options (cf. section ""Optimization" Node"), separate methods for the direct access on the respective elements are created in the generated code, which are called via funcion calls.*
When the **Optimize Direct Access Methods (one level)** *option is activated, the direct access is used instead of separate methods.*
When the **Optimize Direct Access Methods (multiple levels)** *option is activated, this is also true for nested classes.*

The type of pin displayed in the layout depends on the kind of element being enabled. For parameters (including characteristic lines and maps), constants and system constants, as well as send messages, only an output pin can be added. For receive messages, only an input can be added. For variables, as well as send&receive messages, both an input and an output pin can be added.

4.10.2 Dependent Elements

Dependent elements can be parameters and model constants. This means that they are affected by another element.

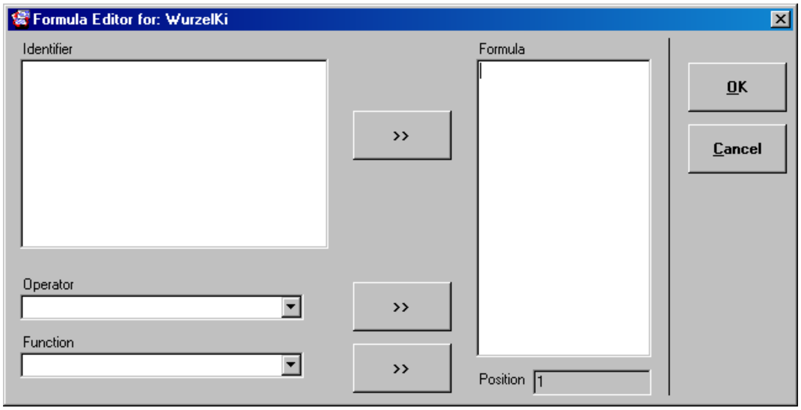
To create the formula for dependent parameters:

- In the Element editor, select the option **Dependent** from the "Dependency" field.

  The **Formula** button is activated.

- Click on the **Formula** button.

  The "Formula Editor" for dependent parameters opens.



Here, you can enter the element dependency as a formula (C-Syntax). The cursor position in the "Formula" field is indicated in the "Position" field.

- Right-click in the "Identifier" field and select **Add** from the context menu.

  This is how you create formal parameters. These parameters are independent of the model.

- Enter a name in the "Identifier" field.

  A list of available operators can be found in the "Operator" combo box.
  A list of available functions can be found in the "Function" combo box.

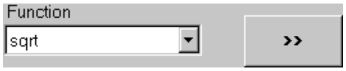
  You can create the formula for the dependency in the "Formula" field using formal parameters, operators and functions.

  There are two ways for doing so:

1. Enter the formula directly in the "Formula" field.

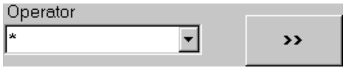
Or

2. Use the **>>** buttons.
   – In the "Function" combo box, select a function.
   – Click on the neighboring **>>** button.

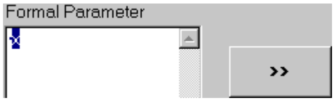


The selected function appears in the "Formula" field, at the cursor position.

   – In the "Operator" combo box, select an operator.
   – Click on the neighboring **>>** button to add the operator to the formula.



   – Highlight a formal parameter in the "Formal Parameter" field.
   – Use the neighboring **>>** button to add the parameter to the formula.



Operators, functions and formal parameters can be inserted in any number or sequence. However, only one can be added at a time.

To edit the formula of a dependent parameter:

You can edit the formula of a dependent parameter at a later state.

- Open the formula editor for the dependent parameter whose formula you want to change.
- Edit the formula as described in the previous passage.
- In the "Formal Parameter" field, right-click on a formal parameter you want to rename and select **Edit** from the context menu.
- Enter the new name.

- In the "Formal Parameter" field, right-click on a formal parameter you want to remove and select **Delete** from the context menu.

Note

The changes made by renaming or deleting formal parameters affect only the "Formal Parameter" field. The formula in the "Formula" field has to be updated manually.

In order to make the dependency of the element in the model effective, the formal parameters must be specified according to the model.

To specify the formal parameters:

- Highlight the element you want to edit in the "Elements" pane.
- Select **Element** → **Edit Data**.

or

- Right-click on the element in the "Elements" pane or the drawing area, and select **Edit Data** from the context menu.

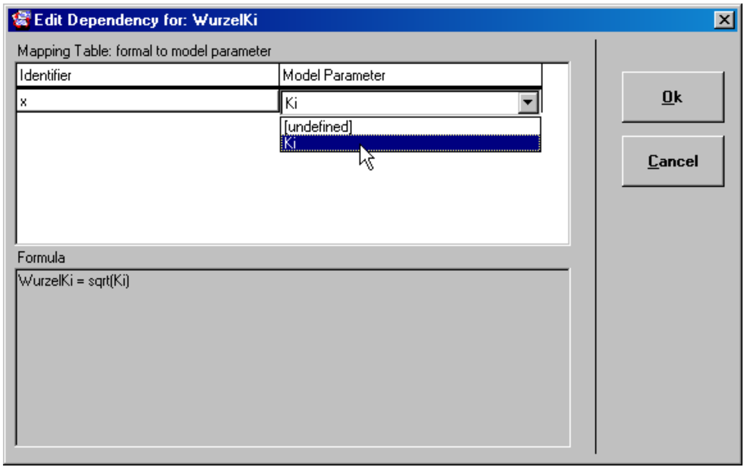
or

- In the Browser view, select the "Data" tab.
  - Highlight the element you want to edit.
  - Press <ENTER>

or

– right-click on the "Data" tab and select **Edit** from the context menu.

The "Edit Dependency" dialog opens.


Edit Dependency for: WurzelKi
Mapping Table: formal to model parameter
Identifier
Model Parameter
x
Ki
[undefined]
Ki
Formula
Ok
Cancel

WurzelKi = sqrt(Ki)

- In the "Model Parameter" column, right-click on the `[undefined]` field.

  A combo box opens that lists the parameters in the component.

- Select one of the listed elements.

  This way you assign the formal parameter a model parameter which determines the value of the dependent parameter.

- Close the "Edit Dependency" window.

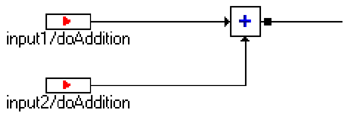
Temporary variables can be used to store the results of statements or operations and avoid multiple execution within the same method or process (see section "Temporary Variables" in chapter "The Kind of Elements" of the ASCET reference guide).

To use temporary variables:

- Right-click on the operator for which you want to use a temporary variable.

- Select **Temporary Variable** from the context menu.

  The operator displays a solid rectangle on the output pin to indicate that the result of the operation is stored in a temporary variable.



- Repeat the command to reverse the setting.

4.11 Editing Data

When specifying a component, you can assign an initial value to each element in your specification. All of these values can be changed at a later stage. You can specify different data sets, i.e. sets of initialization values between which you can toggle, or you can change individual values during experimentation. This section describes the different editors for the various kinds of elements.

Usually a data editor is first called from within the component development environment, e.g. the block diagram editor, to assign a default value to an element. Then the editor can be opened again from within the experimentation environment to calibrate the value of the element during an experiment (see "The Numerical Editor" on page 622). Data editors can also be used to define data sets for components or projects.

Data sets are independent from implementations (cp. chapter 4.12). Nevertheless inconsistencies between both can arise. In the context of a project, the value of an element defined in its data set can exceed the value range defined in the implementation. It is the user's responsibility to use consistent project settings.

Note

At code generation time, the initialization values of basic elements must fit the value ranges defined by their implementations. Otherwise an error message is generated for parameters, a warning for variables.

To open an editor

- In the "Elements" pane of the component editor, highlight the element you want.

- Select **Element** → **Edit Data** on the menu bar.

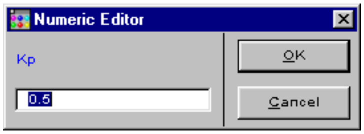
  Similar to opening the element editor (cp. chapter 4.10.1) further possibilities to open the data editor exist.

  Depending on what you selected, one of the editors described below opens.

4.11.1 Editors for Scalar Types

Numerical Editor

The numerical editor shows the value of a numeric element and lets you edit that value.



To edit a value:

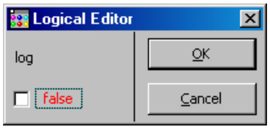
- Enter the value you want.

or

- Click on the numeric display and edit the value as required.
- Click **OK** to import the modification.

Logical Editor

The logical editor shows the value of a logical element and allows you to edit that value.



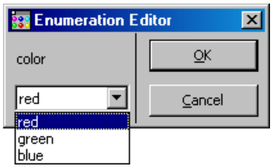
To edit a logical value:

- Tick the option box to set the value to `true`.
- Untick the option box to set the value to `false`.

- Click on **OK** to import your modification.

Enumeration Editor

The Enumeration editor displays the enumerator assigned to the element and allows you to select another enumerator from the current enumeration.



To select an enumerator:

- Click on the combo box.
- Select the enumerator you want.
- Click **OK** to confirm your changes.

4.11.2 The Editor for Combined Types (Table Editor)

Tables are ASCET data structures for describing characteristic lines and maps. The curves in a table are not defined by a mathematical function, but rather by defining individual data points that form the output of particular input values. The maximum number of data points is specified in the specification editor after the table element is selected. ASCET supports one- and two-dimensional tables.

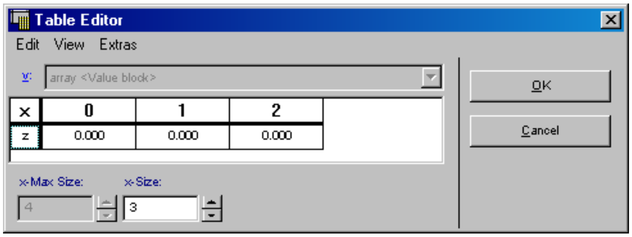
The creation of a table is described in the section "Arrays, Matrices, Characteristic Curves and Maps" on page 211. You can edit the table element from the "Elements" pane. The table editor (also multi editor) contains a menu bar, the "v:" combo box which contains the names of the currently edited tables, a data display field and varying setup fields for table size and interpolation. The data display field shows the sample points (array/matrix: index values) and output values which can be edited according to the type of the edited table. The editors briefly introduced in the following sections are the different forms of the table editor. A more detailed description is given in chapter 6.2.3 "Working with Calibration Windows".

While you can add sample points dynamically during specification and experiments, the actual number of sample points must not exceed the maximum number defined at the creation of the table. The maximum number of values can only be changed in the specification editor with the **Element** → **Max Size** command.

Array Editor

An array is a 1-D table where the values on the x-axis are fixed, i.e. they always start at 0 and are always incremented by one. An array in ASCET corresponds to a one-dimensional array in a conventional programming language.

The array editor is the simplest form of the table editor.



y: array <Value block>

x	0	1	2
z	0.000	0.000	0.000

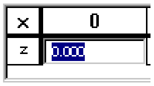
x-Max Size: 4 x-Size: 3

In the data display area, the index values and their associated output values are displayed; two setup fields for the array size are present.

To set up an array:

- Adjust the number of actual x-axis points in the "x-Size" field.

  The specified number of cells is created. During an experiment additional x-axis points can be added, but only up to the maximum size.



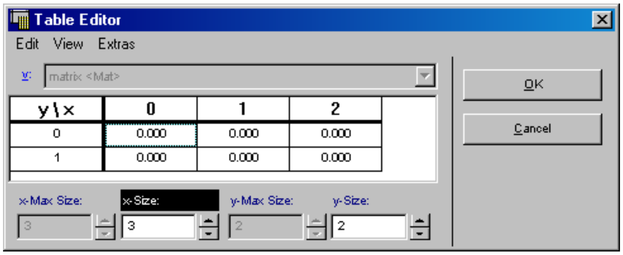
x	0
z	0.000

- In the editor window, double-click on a cell on the z-axis.

  The cell becomes an input field.

- Type in a value and press < ENTER>.
- Repeat for all the other values.

Two-dimensional arrays are also available in the form of matrixes.



y: matrix <Mat>

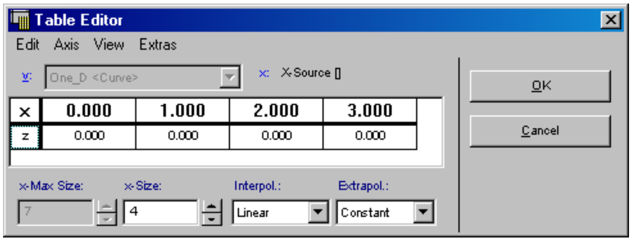
y \ x	0	1	2
0	0.000	0.000	0.000
1	0.000	0.000	0.000

x-Max Size: 3 x-Size: 3 y-Max Size: 2 y-Size: 2

The matrix editor shows the index and output values of both axes and contains four setup fields for the size. Otherwise, it is used exactly as the array editor.

The 1-D Table Editor

A one-dimensional table has one input value and one output value. Internally, a number of sample points is defined and an output value is given for each sample point. If, for instance, you have defined the value 2 at a sample point with the value 0.5, the output of the table will be 2 whenever the input value is 0.5. If the input value falls between two sample points, the result is interpolated between the two values.



x	0.000	1.000	2.000	3.000
z	0.000	0.000	0.000	0.000

x-Max Size: 7 x-Size: 4 Interpol.: Linear Extrapol.: Constant

Thesample points and their respective output values are shown in the data display field; two setup fields for the table size, as well as interpolation setup fields, are present.

The 1-D table editor works in the same way as the array editor. The only difference between the editors is that in the table editor the values on both axes can be edited.

To edit a 1-D table:

- In the block diagram editor, right-click on a 1-D table element and select **Edit Data** from the context menu.

Or

- Highlight the 1D-table element in the "Elements" pane and select **Element** → **Edit Data** on the menu bar.

  The 1-D table editor opens.

- Modify the z-axis values as described on page 461.

- Adjust the number of sample points in the "x-Size" box.

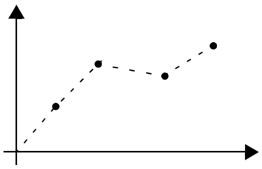
  This creates a table with `n` sample points on the x-axis of the table. By default they are assigned values from `0` to `n-1`.

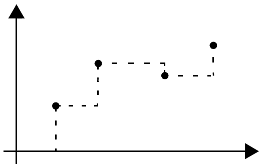
- Select an interpolation mode from the "Interpol." combo box.

  – `Linear` means that the two sample points are connected by a straight line, e.g. if we have a sample point 1 with the value 2 and a sample point 2 with the value 4, the result for an input value 1.5 will be 3.

  – `Rounded` means that the values between two sample points are always the same as the value at the smaller sample point. With the two sample points given above, the output would be 2 for every input value greater than 1 and smaller than 2.

- Click **OK**.


Linear Interpolation


Rounded Interpolation

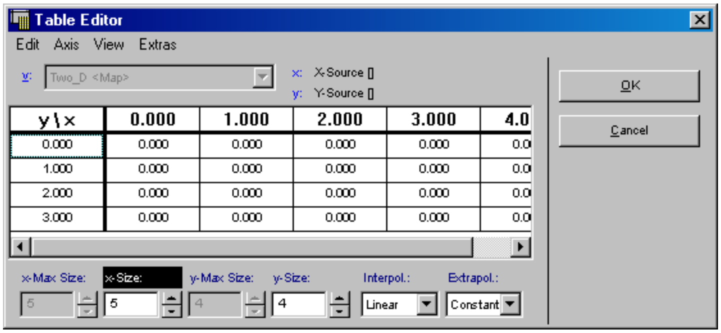
The extrapolation mode is always set to `Constant`. For all the x-values greater than the highest x-value defined in the table, the value returned is the z-value of the highest x-value. For values that fall below the lowest x-value, the z-value of the lowest x-value is returned.

The sample points are shown on the x-axis of the data area, the values are on the z-axis. By default the number of sample points that you entered in the 1-D Table Setup dialog box is created. The default sample points form a range between 0 and the number of sample points minus 1.

The 2-D Table Editor

The difference between the 2-D table and the 1-D table editor is that there are two dimensions of sample points instead of just one. The 2-D table has two inputs and returns one output value for each pair of inputs.

The sample points are shown in the top row and left-most column of the table in the data display area. The 2-D table editor is the most elaborate version of the table editor.

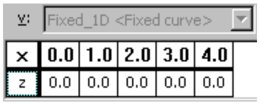


y \ x	0.000	1.000	2.000	3.000	4.0
0.000	0.000	0.000	0.000	0.000	0.0
1.000	0.000	0.000	0.000	0.000	0.0
2.000	0.000	0.000	0.000	0.000	0.0
3.000	0.000	0.000	0.000	0.000	0.0

Fixed Tables

A fixed table is a table where the distance between the axis points cannot be altered. You can select any value for the distance, but it is always the same for all points. If, for instance, the distance between the sample points has been set to 3 and the offset to 4, the first axis point has the value 4, the second one 7, the third one 10 etc.

To set up the axis points:



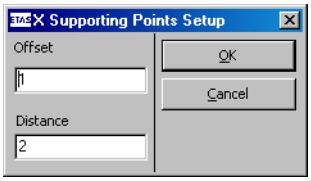
x	0.0	1.0	2.0	3.0	4.0
z	0.0	0.0	0.0	0.0	0.0

- Open the fixed characteristic line you want to edit in the table editor.

  The adjacent image shows a simple example.

- In the table editor, select **Axis** → **X Supporting Point Setup**.

  The "X Supporting Point Setup" window opens.



Offset: 1

Distance: 2

- In the "Offset" field, enter the offset for the first axis point (in the example: 1).

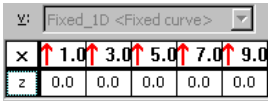
- In the "Distance" field, enter the distance between the axis points (in the example: 2).

Note

*Enter **integer numbers** in both fields, because otherwise errors will occur when working with implementations.*

- Click **OK**.

  The values you entered are accepted, and the axis points are adjusted accordingly.
  The adjacent image shows the updated example.



Y: Fixed_1D <Fixed curve>

x	1.0	3.0	5.0	7.0	9.0
z	0.0	0.0	0.0	0.0	0.0

Or

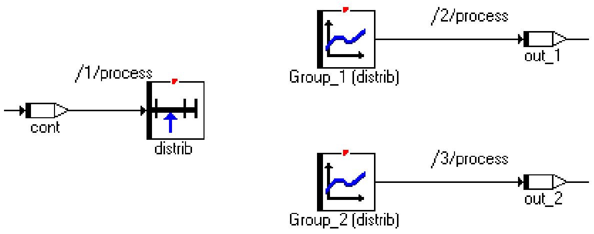
- Click **Cancel** to discard the values and return to the original state.

Fixed tables are available as 1-D tables and 2-D tables. The Y axis points of a 2D fixed table are adjusted via **Axis** → **Y Axis Support Points Setup**. With the exception of the axis point treatment, the editor for fixed tables functions in the same way as the table editor.

Group Tables

Group tables are tables that share the same distribution of axis points but have different return values. This means that it is possible to obtain different return values for the same input. The distribution of axis points and the individual group tables are specified as separate elements; the former are called distributions.

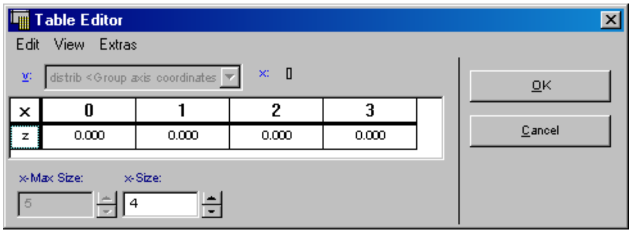
Axis points, but no return values, are specified in a distribution. Once a distribution has been defined, group tables using that distribution can be specified. In a block diagram, a distribution has only an input, whereas group tables only have an output, as shown below:



Group tables are available as 1-D tables and 2-D tables.

Group Table Editor: The group table editor works in the same way as the table editor. When creating a group table, you are prompted for a distribution. This distribution is then assigned to the table, i.e. all the sample points defined in the distribution are also defined for the group table. It is not possible to make any changes to the x axis in this editor; however, it is possible to switch to another distribution from the specification editor using **Element → Edit Distribution**. Assigning values on the z-axis works in the same way as in the 1-D table editor.

Distribution Editor: The distribution editor works in the same way as the array editor. The main difference is that the sample point axis values are added to the z axis on the table. The x-axis serves for navigation purposes only and cannot be changed. None of the commands for changing the x-axis are available in this editor.



y: distrib <Group axis coordinates   x: 0

x	0	1	2	3
z	0.000	0.000	0.000	0.000

x-Max Size: 5   x-Size: 4

4.11.3 Data Sets

Every component or project can have a number of data sets. Data sets determine the initial values of the elements, i.e. parameters and variables, and of the components. A data set contains one initialization value for each element of the component or project. Therefore, data sets define variants of components and projects.

Data sets are handled in the same way as for components and projects. Each component data set includes the initial values for all the basic elements used in that component. For complex elements, which have their own data sets, the component data set has a reference to one of the data sets of the complex element.

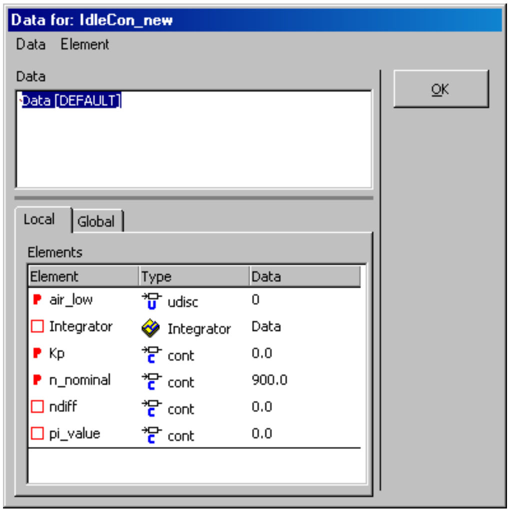
Each component can have several data sets. When a component is first created, a default data set with the name `Data` is created with it.

To view data sets

- In the specification editor, select **Component** → **Edit Data** to view the data for the current project or component.

  The "Data for: <*Component*>" dialog window opens.

  The names of all the project data sets are displayed in the "Data" pane.
  All components included in the project or component are displayed in the "Elements" pane, together with elements. With basic elements, the value is specified after the element name and type. With complex elements the name of the referencing dataset is displayed.



Data [DEFAULT]

Element	Type	Data
air_low	udisc	0
Integrator	Integrator	Data
Kp	cont	0.0
n_nominal	cont	900.0
ndiff	cont	0.0
pi_value	cont	0.0

- Select the data set you want to view in the "Data" pane.

  The values for the data set selected are displayed in the "Elements" pane.
- Click **OK**.

The data set currently selected, for which the values are shown, becomes the active data set when you close the data editor. You can also open the dataset editor from the browser by selecting a referenced component in the "Data" tab.

To create/copy a new data set:

- Select **Data** → **Add**.

  A new data set is created. All elements have a default value. This is either `0.0` or `true` for basic elements, or the corresponding default data set for complex elements.

Or

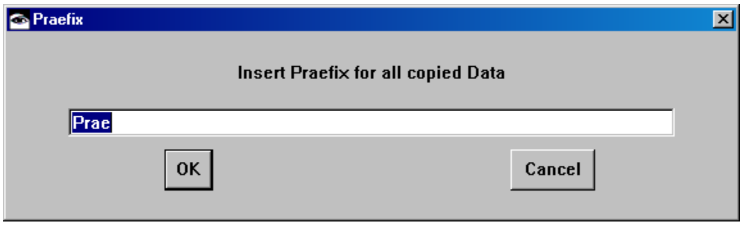
- Select **Data** → **Copy** → **Flat.**

  The active data set is copied, i.e. the newly created data set contains the same values. If there are references to other data sets, they are copied as well.

Or

- Select **Data** → **Copy** → **Recursive**.

  An input window opens.



Praefix

Insert Praefix for all copied Data

Prae

- Enter a prefix for the names of the copied datasets and click **OK**.

  The active data set is copied, and recursive copies made of all the referenced data sets, i.e. the copies of those data sets are also recursive. The new data set has references to the copies of the referenced data sets.

Note

This process only works for local variables.

To delete a data set:

- In the "Data for: <*Component*>" window, select the data set you want to delete.
- Select **Data** → **Delete**.

  The data set is deleted. If other data sets have referenced that data set, they now reference the data set that becomes active after deletion. The default data set cannot be deleted.

To rename a data set:

- Select the data set you want to rename
- Select **Data** → **Rename**.
- Type in the new name and press <ENTER>.

When a data set in the "Data for: <*Component*>" dialog box is selected, it becomes the active data set. If you close the component or project editor and then open it again, the default data set will however be active again. There is always one default data set associated with each component and project; initially it is the one that is created automatically when the component or project is created.

To make a data set the default:

- Open the "Data for: <*Component*>" dialog.
- Select the data set you want to make the default.
- Select **Data** → **Become Default**.

  The selected data set becomes the default. Whenever the component or project is opened, this data set will be active.

If a data set contains references to other data sets, it is possible to browse the data sets those references point to.

To browse a data set:

- Select the data set you want to browse.
- Select the complex element (component) with the data set you want to browse.
- Select **Element** → **Edit**

or

- double-click the complex element you want in the "Elements" pane.

  The "Data for: <*Component*>" dialog window for the referenced data set opens.
- Use this procedure recursively to browse through an entire data set hierarchy.

To edit data in a data set:

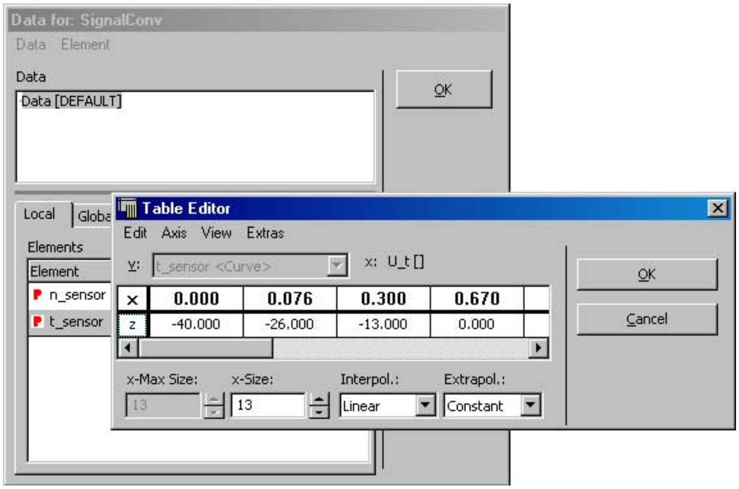
You have several possibilities to edit the data of sindividual elements.

1. Open data editor

- Select the basic element you want to edit.

  If you want to edit a basic element in a referenced data set, you have to open it first as described above.
- Select **Element** → **Edit**.

  The appropriate data editor opens, e.g., the 1-D table editor opens if the element is a characteristic line.



x	0.000	0.076	0.300	0.670
z	-40.000	-26.000	-13.000	0.000

x-Max Size: 13
x-Size: 13
Interpol.: Linear
Extrapol.: Constant

2. Copy data

- Select the basic element whose data you want to copy.

- Select **Element** → **Copy Data To Buffer**

or

- select **Copy Data to Buffer** from the context menu.

  The element data are copied to the database clipboard.

- Select the basic element you want to copy the data to.

Note

You can exchange data only between elements of the same kind. For composite types (arrays, matrices, tables), the current size has to be the same, too.

- Select **Element** → **Paste Data From Buffer**

or

- select **Paste Data From Buffer** from the context menu.

  The data are copied from the database clipboard to the element.

The data set of a project or a component can be exported separately from the functional specifications. With this feature the data sets and the specification can be developed in parallel.

To export a data set:

- In the "Data for: <*Component*>" dialog, select the data set you want to export.
- Select **Data** → **Export**.

  The Windows file selection dialog box opens.

- Type in a data set name and press <ENTER>.

  The data set selected is exported.

The same commands are also available in the main menus of the specification editors (**Component** → **Export Data**) and the project editors.

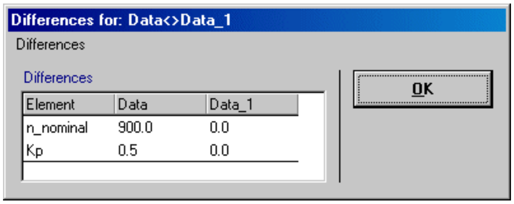
To show the differences between two data sets:

- Select two of the of the component or project data sets in the "Data" pane.

  You can select multiple data sets by clicking on them while holding down the <CTRL> key.

- Select **Data** → **Show Differences** → **Flat**.

  The data sets are compared on the first level, i.e. only the values for the basic elements contained in the project or component are compared. Referenced data sets are not taken into account. The "Differences" dialog box opens showing the differences in the data sets. The differences for scalar elements are shown directly, non-scalar elements with different data are only listed.



Differences for: Data<>Data_1

Differences

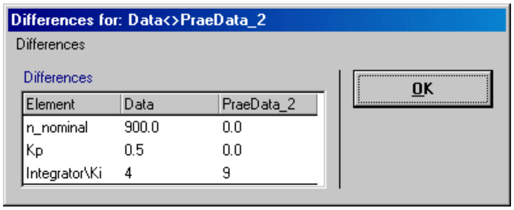
Differences

Element	Data	Data_1
n_nominal	900.0	0.0
Kp	0.5	0.0

Or

- Select **Data** → **Show Differences** → **Recursive**.

  The data sets are compared recursively, i.e. independent of the data sets referenced, the values of the basic elements in all referenced components are compared.



Differences for: Data<>PraeData_2

Differences

Differences

Element	Data	PraeData_2
n_nominal	900.0	0.0
Kp	0.5	0.0
Integrator\Ki	4	9

- In the "Differences" window, select **Differences** → **Copy To Clipboard** to copy the results of the comparison to the clipboard.

  They can now be pasted into another application.

It is possible to write the data from a table or an array to a file and to read it in again. The data is written in tab-delimited ASCII format.

To write array or table data to a file:

- In the "Elements" pane of the data editor or the specification editor, select a table or array.
- Select **Element** → **File Out Data**.

  The Windows file selection dialog box opens.
- Select a path and a filename.
- Click on **Save**.

  The data from the table or array selected is written to the file.

To read the data for an array or a table from a file:

- In the "Elements" pane, select the table or array.
- Select **Element** → **File In Data**.

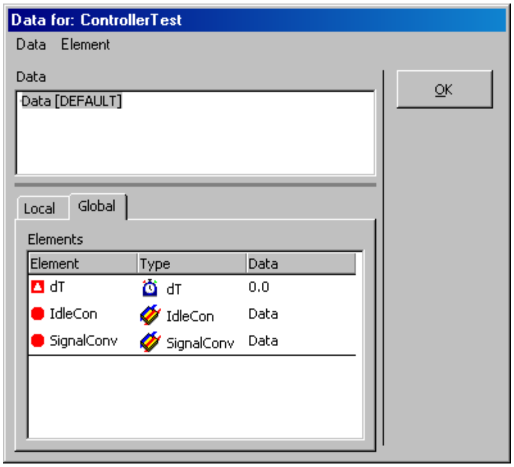
  The Windows file selection dialog box opens.
- Select the file that contains the data you want to read.
- Click on **Open**.

If the file format does not match the selected element, the data set is left unchanged and a message box appears. This happens, for instance, if the sample points are not given in monotonous rising order. If the file contains missing numbers, these are treated as zeroes (for instance if the number of sample values does not match the number of sample points). If the file contains invalid characters, these are interpreted as zero. The size of the element is adjusted to the data that is read in.

There is one major difference between components and projects, namely that projects have global elements as well as the components they reference. In contrast to the data for the components, where a project can have several data sets, there is only one data set for the global elements.

To work with the global elements data set:

- In the project editor, select **Component** → **Edit Data** to open the data editor dialog window.
- Click on the "Global" tab of the dialog to view the data for the global elements in the project.



Element	Type	Data
dT	dT	0.0
IdleCon	IdleCon	Data
SignalConv	SignalConv	Data

Even if more than one data set is shown in the "Data" pane, you cannot select ot edit them.

An additional tool set for data exchange is available in ASCET. Users can perform flat and recursive file in and file out operations for the parameters in data sets using various filter and data exchange options and file formats.

To set the data exchange options:

- In the Component Manager, open the ASCET options window.
- In the "Data Exchange" node, adjust the desired options.

  The data exchange options are described in section "Data Exchange Options" on page 65.

Once you have set the data exchange options for your system, you can use the data exchange tool set to import and export data sets. The tool set can be accessed through the corresponding menu choices in the data set editor.

To use the data exchange tool set:

- In the data set editor, select the data set you want to edit.
- Select **Data** → **File Out Recursive** to file out the selected data set and all referenced data sets.
- Select **Data** → **File Out** to file out the selected data set.
- Select **Data** → **File In Recursive**, to file in a data set with all referenced data sets.
- Select **Data** → **File In** to file in a data set.
- Select **Data** → **Show Data File** to view a data file.
- Select **Data** → **Show File In Log File** to view the log file for read processes.
- Select **Data** → **Show File Out Log File** to view the log file for write processes.

4.12 Editing Implementations

Three implementation editors are available, one for components and projects, one for basic elements – i.e. variables, parameters, and system constants – and one for methods and processes. Depending on the kind of the elements, fields are deactivated, or additional tabs appear, in the implementation editor for basic elements, however, the structure is always the same.

Implementations are independent from data sets (cp. chapter 4.11). Nevertheless inconsistencies between both can arise. In the context of a project, the value of an element defined in its data set can exceed the value range defined in the implementation. It is the user's responsibility to use consistent project settings.

Note

At code generation time, the initialization values of basic elements must fit the value ranges defined by their implementations. Otherwise an error message is generated for parameters, a warning for variables.

For literals, the most suitable implementation is derived automatically by the code generation, so it does not have to be specified. Their implementation is derived from the first assignment made to them. The same is true for constants.

4.12.1 Implementations of Components/Projects

You can open the implementation editor for components/projects

- from the Component Manager (cf. section "To edit the implementation:" on page 86),
- from any specification editor

or—for included components—

- from the implementation editor of the containing component,

and edit the values for the current component or project.

To open the implementation editor of a component:

1. edited component/project
   - In the specification editor, select **Component** → **Edit Implementation** to edit the implementation for the current project or the current component.
2. included component (specification editor)
   - In the "Elements" list, select an included component.
   - Select **Element** → **Implementation**

   or

   - select **Edit Implementation** from the context menu

   or

   - use the Browser view of the specification editor:

     – Click on the **Browse** tab.
     – In the Browser view, open the implementation editor for the included component as described on page 86.
3. included component (implementation editor)
   - In the "Elements" list, select an included component.

- Select **Element** → **Edit**

or

- double-click on the component

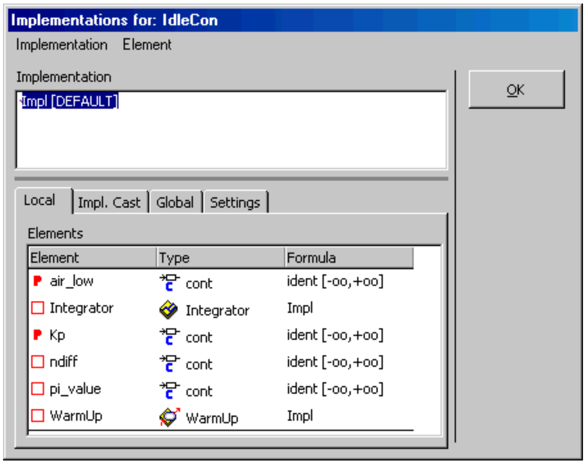
or

- select **Edit** from the context menu

or

- press the <RETURN> key.

The implementation editor for the selected component opens.



Impl [DEFAULT]

Element	Type	Formula
air_low	cont	ident [-oo,+oo]
Integrator	Integrator	Impl
Kp	cont	ident [-oo,+oo]
ndiff	cont	ident [-oo,+oo]
pi_value	cont	ident [-oo,+oo]
WarmUp	WarmUp	Impl

The names for all implementations defined for this component or project are shown in the "Implementation" pane.

All the components and local elements included in the project or component are displayed in the "Locals" tab; the "Impl. Cast" tab contains the implementation casts (see "Implementation Casts" in the ASCET reference guide). The "Globals" tab lists global elements. With basic elements, the formula is specified after the element name and type.- With complex elements the name of the referenced implementation is displayed.

The "Settings" tab contains several setup options.

To select an implementation:

- Select the implementation you want to edit in the "Implementation" pane.

  The formulas for the implementation selected are displayed in the "Elements" pane.

  You can open the implementation editor for an element or included component by double-clicking on an entry in the "Elements" pane.
- Click **OK** to close the window.

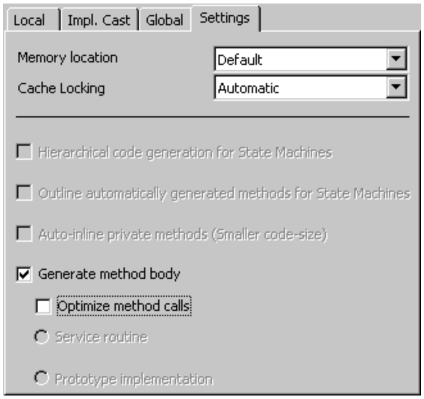
All the operations available for data sets are also available for implementations and work as described in chapter "Data Sets". You can specify implementations for local and global elements, as well as implementation casts, by selecting the corresponding tab of the implementation editor.

To adjust the implementation settings:

Note

The "Settings" tab has no meaning until you select the experiment `Object Based Controller Implementation` *in the build options.*

- Select the "Settings" tab.



Memory location: Default
Cache Locking: Automatic
[ ] Hierarchical code generation for State Machines
[ ] Outline automatically generated methods for State Machines
[ ] Auto-inline private methods (Smaller code-size)
[✓] Generate method body
[ ] Optimize method calls
○ Service routine
○ Prototype implementation

Only the "Memory location" and "Cache Locking" combo boxes are available for all components (classes, modules, projects), the options are deactivated for modules and projects.

- In the "Memory location" combo box, select the memory area where the data structure for the component is to be stored.
  The memory class for the code is defined in the implementation editor of the respective method or process (cf. chapter 4.12.8 on page 500).
- When you are editing the implementation of a class, make sure that the **Generate method body** option is activated so that code is generated for the class.

  The options **Hierarchical code generation for State Machines**, **Outline automatically generated methods for State Machines** und **Auto-inline private methods (Smaller code-size)** are available only for state machines. Their usage is described in "To activate/deactivate outlining of actions/conditions:" and "To activate/deactivate auto-inlining:".

Cache locking is described in the ASCET-RP user's guide, the other settings are described in the ASCET-SE user's guide.

4.12.2 Implementation of Scalar, Non-logical Elements

The implementation of non-logical elements describes the transformation from an infinite model domain (either `continuous` or `discrete`) to a finite implementation domain. Therefore the range for the values in the model has to have interval limits. Additionally, a linear formula describing the transformation from the physical to the implemented representation has to be defined.

For the model type `continuous`, which has an infinitely fine resolution, the formula determines the quantization in the implementation domain using special fixed point arithmetic. The quantization is the reciprocal of the gradient of the linear formula since the implementation is assumed to be in integer arithmetic.

To open the implementation editor for basic elements:

1. from the Component Manager:
   - In the "1 Database" list, select a component or project.
   - In the d "3 Contents", "Implementation" tab, select the basic element whose implementation you want to edit.
   - Select **Component** → **Edit Item**

   or
   - double-click on the element

   or
   - select **Edit** from the context menu

   or
   - press the <RETURN> key.
2. from the specification editor:
   - In the "Elements" list, select the basic element whose implementation you want to edit.
   - Select **Element** → **Edit Implementation**

   or
   - select **Edit Implementation** from the context menu

   or
   - use the Browser view of the specification editor:

     – Click on the **Browse** tab.
     – In the Browser view, open the implementation editor as described on page 86.
3. from the implementation editor of a component/project:
   - In the "Elements" list, select the basic element whose implementation you want to edit.
   - Select **Element** → **Edit**

   or
   - double-click on the element

   or

- select **Edit** from the context menu

or

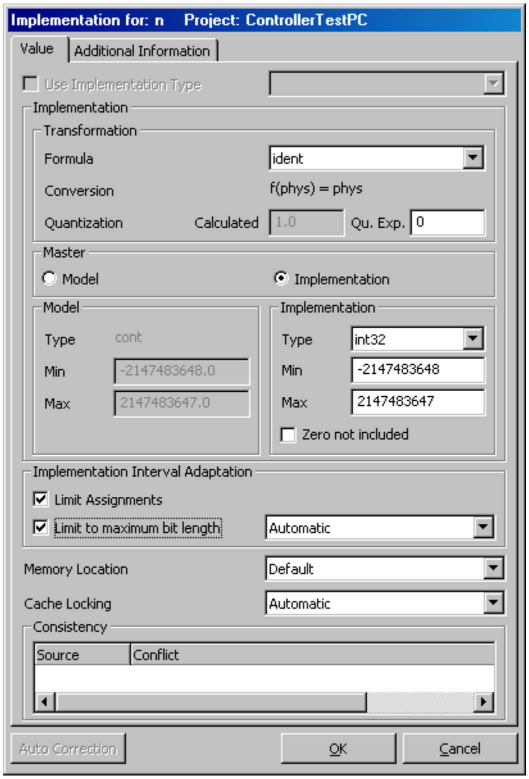
- press the <RETURN> key.

The implementation editor for the basic element opens. The model type is given by the element.

Similar to the element editor (cp. chapter 4.10.1), further possibilities exist to open the implementation editor for basic elements.

For newly created elements, the system does not assume the assignment of an implementation type defined in the project context (cf. "Implementation Types" on page 430), it assumes the assignment of a local implementation. For the `cont` and `sdisc` model types, the implementation data type selected in the "Implementation" node of the options window (see page 52) is selected. For the value ranges, the maximum limits for the type are set.

The implementation editor for scalar, non-logical elements looks like this:


Implementation for: n   Project: ControllerTestPC

Value | Additional Information

☐ Use Implementation Type

Implementation

Transformation

Formula: ident

Conversion: f(phys) = phys

Quantization: Calculated 1.0   Qu. Exp. 0

Master

( ) Model   (●) Implementation

Model		Implementation	
Type	cont	Type	int32
Min	-2147483648.0	Min	-2147483648
Max	2147483647.0	Max	2147483647
		☐ Zero not included	

Implementation Interval Adaptation

☑ Limit Assignments

☑ Limit to maximum bit length: Automatic

Memory Location: Default

Cache Locking: Automatic

Consistency

Source	Conflict

The "Implementation for" dialog window offers the possibility to select a predefined implementation type (see "Implementation Types" on page 430) via the **Use Implementation Type** option and the adjacent combo box, or to specify an individual implementation in the "Implementation" field. In each case, the limiting behavior ("Implementation Interval Adaptation" field) and the memory area in which the element is located ("Memory Location" combo box, only relevant for micro-controller targets) are set individually. The "Consistency" field displays warnings and error messages.

In the "Additional Information" tab, you can enter information relevant to individual code generators, which is only evaluated where applicable. The exact nature of the information you can enter here depends on your target and code generator.

For scalar basic elements, the **Use Implementation Type** option is deactivated by default; the adjacent combo box is grayed out. You do not have to change anything here to specify an individual implementation.

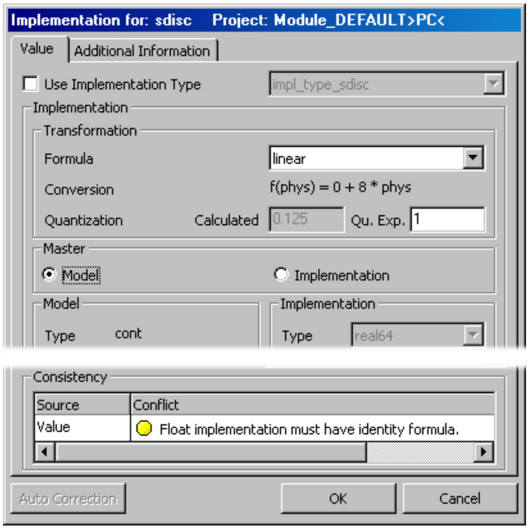
To select a formula:

- Select the transformation formula from the "Formula" combo box. All the formulas included in the project are available.

  The quantization in the "Calculated" field is determined automatically based on the formula. If the implementation data type `real*` is selected, the quantization 1is displayed.

  In addition, a quantization can be entered in an extra field ("Qu. exp."), which is used exclusively for the `Quantized Physical Experiment`.

If the variable has the model data type `cont` and the implementation data type `real32` or `real64`, or if the variable has a model data type different from `cont`, only the identity formula should be selected because only this formula is supported by the code generation. If you select another formula, a warning is displayed.



Implementation for: sdisc   Project: Module_DEFAULT>PC<

Value | Additional Information

- [ ] Use Implementation Type: impl_type_sdisc

Implementation

Transformation

Formula	linear
Conversion	f(phys) = 0 + 8 * phys
Quantization	Calculated 0.125 Qu. Exp. 1

Master: [✓ Model] Implementation

Model — Type: cont

Implementation — Type: real64

Consistency

Source	Conflict
Value	● Float implementation must have identity formula.

- To switch to the identity formula, select `ident` from the "Formula" combo box.
- Click **OK** to close the implementation editor.

Or

- Click **OK** without selecting the identity formula.

  The selected formula is accepted.

Or

- Click **Cancel** to close the dialog and restore the original settings.

When you edit an element of `sdisc` or `udisc` type for which a formula other than the identity was selected in a previous ASCET version, an error message appears.

Note

Models from earlier ASCET versions can used unchanged. During code generation however warnings are generated if they contain formulas for discrete elements.

When you make changes in a formula, or recursively replace a formula in a project implementation, you can have all your implementations updated automatically to match the new value ranges and data types. For information on replacing formulas and updating implementations see section "Global Changes in Implementations" on page 429.

To set the master page of an implementation:

ASCET can either use the model or the implementation page of an implementation as the starting point for automatic updates. The starting point is set with the options in the "Master" field. You can set the default master page in the "Implementation" node of the options window (see "Implementation Options" on page 52).

- In the implementation editor, click on the **Model** option to specify that the settings on the "Model" side are used as the starting point for updating an implementation.

  The "Min" and "Max" fields in the "Model" pane are activated.

  The model data type cannot be edited, it is determined by the element and merely displayed in the "Type" field.

The fields in the "Implementation" pane are disabled, except for the **Zero not included** option; they are solely used as displays.

Or

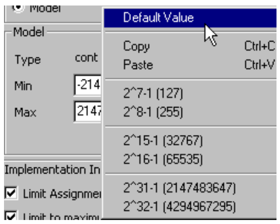
- In the implementation editor, click on the **Implementation** option to specify that the settings on the "Implementation" side are to be used as the starting point for updating an implementation.

  The fields in the "Implementation" pane are activated.
  The fields in the "Model" pane are disabled and serve as displays only.

To specify an implementation (master: model):

The **Model** option should be selected as master if physical model properties are the main reason for selecting the implementation. In this case, proceed as follows to specify the implementation.

- In the "Min" and "Max" fields, enter the limits of the physical interval.



- To enter the default limits for the model data type, right-click in one of the fields and select **Default Value** from the context menu.

  The maximal values for the respective model data type are inserted.

Type	Min.	Max.
`cont`	`-oo`	`-oo`
`sdisc`	`-2147483648`	`2147483647`
`udisc`	`0`	`4294967295`

  The implementation values are updated automatically, according to the values you entered and the formula.

To specify an implementation (master: implementation):

The **Implementation** option should be selected as master if certain code properties (e.g., a required bit width) are the main reason for selecting the implementation. In this case, proceed as follows to specify the implementation.

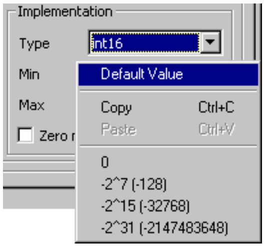
- In the "Type" combo box, select the implementation data type.

  The combo box contains all available types for the element.

- In the "Min" and "Max" fields, enter the limits of the interval.

Note

When you selected a `real*` *implementation data type, the code generation ignores the limits you entered and uses ±oo for model and implementation.*



- To enter the default limits for the implementation data type, right-click in one of the fields and select **Default Value** from the context menu.

  The maximal values for the respective type are inserted.

Type	Min.	Max.
`real64`[a]	`-oo`	`-oo`
`real32`[a]	`-oo`	`-oo`
`int32`	`-2147483648`	`2147483647`
`uint32`	`0`	`4294967295`
`int16`	`-32768`	`-32767`
`uint16`	`0`	`65535`
`int8`	`-128`	`127`
`uint8`	`0`	`255`

a: model data type cont only

The model values are updated automatically.

The values inserted for the model or implementation are checked for consistency, together with the formula.

- If the model side is the master, the min. and max. values on the implementation side are adapted automatically using the formula.

  If the input causes a larger interval in the "Implementation" column than the allowed range of the selected data type, the data type is adjusted automatically (exception: `real*` is selected as implementation data type). In doing so, the first data type suitable for the interval and larger than or equal to the minimum type set in the ASCET options (cf. "Implementation Options" on page 52) is selected.

  Example: If the type `uint8` has been entered in the "Implementation" pane, and the interval `[-100..255]` results from the "Model" settings,

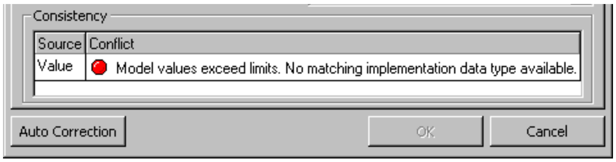
  – the type `int16` is chosen instead, provided the "Minimum cont Data Type" is `int16` or smaller;

  – the "Minimum cont Data Type"is chosen, if it is larger than `int16`.

  The "sizes" of the data types are assumed according to the following sequence:

  `int8` → `uint8` → `int16` → `uint16` → `int32` → `uint32`

  If even `uint32` is too small, an error message is issued (even though the element is of model type `cont` and the implementation data types `real32` and `real64` are available).

  Consistency

Source	Conflict
Value	🔴 Model values exceed limits. No matching implementation data type available.

  When you answer the error message with **Auto Correction**, a value is inserted on the master page which is chosen to ensure that the *implementation side*, after the automatic update, uses the maximum value for one of the 32 bit integer formats. If you click on **Cancel**, the dialog is closed and the original settings are restored.

  If a smaller interval is selected on the "Model" side later, the implementation data type is scaled down again, if appropriate. However, it will never be smaller than the "Minimum ... Data Type".

- If the implementation side is the master, the values on the model side are adapted automatically using the formula.

  When creating an element, it receives the default type set in the ASCET options (cf. "Implementation Options" on page 52) at first. If the user enters a bigger interval in the "Implementation" column than the data type allows, the data type is adjusted automatically. In this case, the smallest data type fitting the interval is selected. In contrast, however, *no* automatic adjustment is done if the user enters an interval, which would permit smaller data types.

Example: If for a `cont` element the type `uint8` is selected in the "Implementation" pane and

– the user enters the interval `[-100..255]`, the `int16` type will be selected automatically instead.

– the user enters the interval `[0..100]`, the `uint8` type will be kept.

For the "sizes" of the data types, again the following order is assumed:

`int8` → `uint8` → `int16` → `uint16` → `int32` → `uint32`

If even `uint32` is too small, the maximum value for one of the 32 bit integer formats will be used automatically without generating an error or a warning.

- If the new value conflicts with one of the other values, for instance if the new minimum value is greater than the maximum value, an error message will be shown, which you can acknowledge with **Auto Correction** or **Cancel**.
- If the implementation is the master, and if you edit an element with model data type `udisc` and implementation data type `int*`, an error message is shown in the "Consistency" field when you leave the implementation editor after entering negative values for the implementation interval.

```
Model type udisc minimum cannot store
     <negative value>.
```

- If the computed min and max values differ from the stored values, the system assumes the current formula as cause of the error. The following error message is shown:

```
Values for min/max are not consistent with
     current formula.
```

This happens, e.g., when you edit a formula in the project, but do not update the implementations afterwards (cf. page 429).

You can always make the following settings, regardless of the master page.

To exclude Zero from the implementation interval:

By default, the code generation assumes that the implementation interval can include zero, i.e. the **Zero not included** option is deactivated. It is checked whether the denominator of a division contains zero. If required, C code is generated that prevents a possible division by zero at runtime (cf. page 417). You can switch off this check.

- When the implementation interval does *not* contain zero, activate the **Zero not included** option.

  The code generation now assumes that this variable cannot become zero. If the variable is used as the denominator of a division, no check is performed.

Note

A division by zero can bring about critical states. If you deactivate the automatic check, it is your responsibility to avoid divisions by zero through appropriate measures in the ASCET model.

When working with older models, **Zero not included** is deactivated for all elements not connected to an implemented operator. The handling of operator implementations from previous ASCET versions is described in chapter 4.12.11.

To set the limitation:

A calculation might result in values outside the interval limits for a variable. The limiter takes into account the interval limits of a variable for all assignments to this variable, i.e. the code generator creates a limiting code. The limits obviate the need for manual limitation of individual variable values.

Note

This option, **Limit Assignments***, is activated by default for newly created elements. When you set the model type to* `udisc` *(cf. page 451), the option is deactivated automatically.*

If the implementation data type `real64` *or* `real32` *was selected for an element, the option* ***cannot*** *be edited.*

- Activate the **Limit Assignments** option.

  You have thus determined that the value range of a variable, defined by "Min" and "Max", is considered when the code generator makes assignments.

Note

Code generation with `Physical Experiment` *(cf. page 412) ignores the option.*

  A variable value is limited by the value range. If the assigned value is out of range, the relevant "Min" or "Max" limit value is used.

Or

- Deactivate the **Limit Assignments** option.

  You have thus determined that the defined value range of a variable is not considered when the code generator makes assignments. In this case, the value of a variable is not limited by the value range, the maximum limit is determined by the implementation data type (`int8`, `int16`, etc.).

Instead of the **Limit Assignments** option, the implementation editor of previous ASCET versions contained the "Use Limiters" field with the options **Yes** and **No**. When you are working with older models, **Limit Assignments** is set according to the settings in that field.

To set the overflow handling:

You can define the overflow behavior. Most options are available in connection with arithmetic services (see chapter 4.14 "Arithmetic Services").

- Activate the **Limit to maximum bit length** option when the result of an operation shall be limited in case of overflow.

  When the result value range exceeds the range specified in the "Integer Arithmetic" node (cf. page 417), code is generated that avoids the overflow.

  The combo box next to the option is activated.

- From the combo box, select `Reduce Resolution` when the resolution can be reduced upon limiting.

  This option avoids an overflow by right-shifting the inputs, if necessary. The shift is determined automatically.

- Select `Keep Resolution` when the resolution shall not be reduced upon limiting.

  If arithmetic services are activated, this option avoids an overflow by using limiting services, if necessary.
  If not, code generation is aborted with an error message.

- Select `Automatic` to make overflow handling dependent on the use of arithmetic services.

  If this is enabled, the "Keep resolution" method is used in case of overflow limiting. If not, the "Reduce resolution" method is used.

When working with older models, **Limit to maximum bit length** is set according to the former "Use Limiters" field. `Automatic` is selected for all elements. The handling of operator implementations from previous ASCET versions is described in chapter 4.12.11.

To select a memory location:

- In the "Memory Location", select the memory area where the element is located.

  The available selection depends on the target.

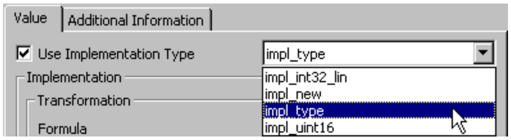
Using Implementation Types

Instead of the individual implementation, you can also assign a predefined implementation type. For details on how to create this, refer to the section "Implementation Types" on page 430; this section looks at how to use it.

The implementation types from the current project context are available to you. That is the default project (see chapter 4.8.1), if you are editing the implementation from a component editor, or the project from which you are editing the implementation.

To assign an implementation type:

- Open the implementation editor for a basic element.
- Actibvate the **Use Implementation Type** option.



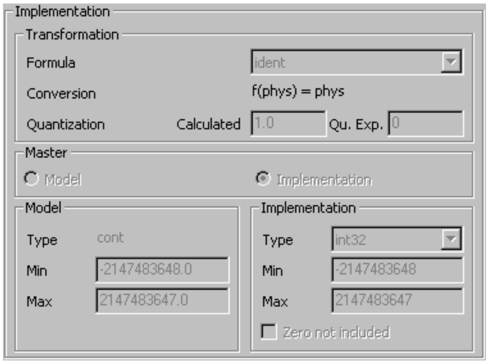
The adjacent combo box is now available. It contains all available implementation types.

- Select an implementation type.

Note

You can only assign an implementation type for variables of the same model type, i.e. an implementation type with the model type `cont` *can only be assigned to continuous variables, not those of type* `sdisc` *or* `udisc`*.*

The settings for model and implementation are accepted. These values cannot be changed; all fields in the "Implementation" area are locked for entries.


Implementation
Transformation
Formula: ident
Conversion: f(phys) = phys
Quantization: Calculated 1.0 Qu. Exp. 0
Master
Model
Implementation
Model
Type: cont
Min: -2147483648.0
Max: 2147483647.0
Implementation
Type: int32
Min: -2147483648
Max: 2147483647
Zero not included

- Set the limitation as described on page 489.

- Select the memory location for the element in the "Memory Location" dropdown list.

Implementation types are referenced by the implementations of basic elements. If it is intended to use a copy of an implementation type, e. g. for a subsequent modification, an implementation type can be first activated and then de-activated again. In this case, the settings of the implementation type are adopted by the implementation of the basic element.

Note

As soon as an implementation type is activated for a basic element, the previously used implementation gets lost.

4.12.3 Implementations of Method- and Process-Local Variables

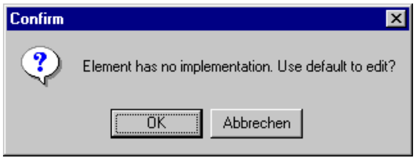
Method- and process-local variables can be implemented automatically or explicitly. Default is automatic implementation; the implementation is derived from the first variable assigned to the method-/process-local variable. If you do not want to use the default, proceed as follows:

To implement a method-/process-local variable:

- In the implementation editor of the component, select the "Local" tab.
- In the "Elements" field, select the method-/process-local variable.

  The menu item **Element** → **Use automatic Implementation** is marked and grayed out if no explicit implementation was specified.
- Open the implementation editor for the method-/process-local variable.

  When automatic implementation is activated, a warning window opens:



Element has no implementation. Use default to edit?

- Confirm by clicking **OK**.

  The implementation editor opens.

- Implement the variable as described in "Implementation of Scalar, Non-logical Elements" on page 479.

  The menu item **Element** → **Use automatic Implementation** is now available. It is no longer marked.

The automatic implementation of a method-/process-local variable can only be reactivated in the implementation editor of the component.

To activate automatic implementation:

- In the "Elements" field, select the local variable.
- Select **Element** → **Use automatic Implementation**

or

- right-click onto the variable name and select **Use automatic Implementation** from the context menu.

  With that, the automatic implementation of the method-/process-local variable is activated. The menu item **Element** → **Use automatic Implementation** is marked and grayed out.

Automatic implementation is deactivated by explicitly implementing the variable.

4.12.4 Implementations for Temporary Variables

Temporary variables can be specified at the outputs of operators and complex model elements. For these temporary variables, the code generator determines the implementation automatically: when a temporary variable is assigned an implemented quantity for the first time, it obtains the corresponding conversion formula and value range. The implementation data type is chosen so that it is appropriate for the conversion formula and value range.

Note

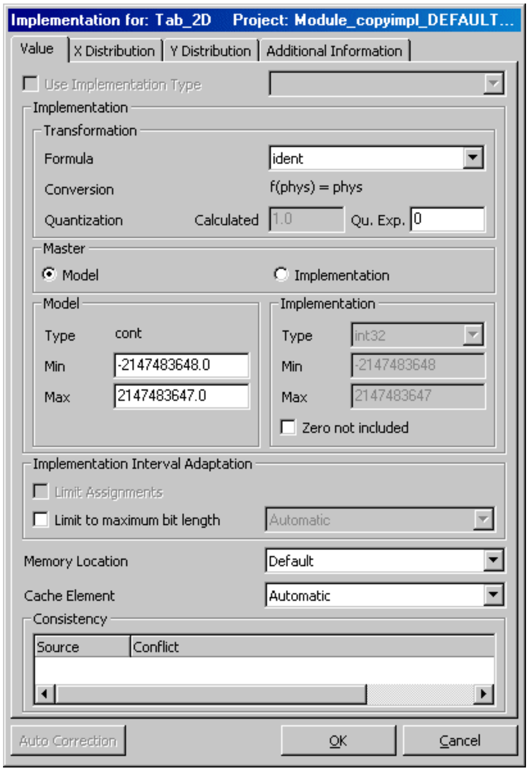
The insertion of temporary variable in a mathematical expression does not affect the generation of mathematical operations for this expression. Temporary variables should not be used in different branches of the control flow (e.g., in the branches of an `If` statement). The result and the implementation (e.g., quantization) may be different for the separate branches. This could cause serious arithmetical errors in the generated code.

4.12.5 Implementations of Arrays, Matrices, and Tables

Arrays and matrices have implementations which are defined like that of a scalar element. A 1-D table has two implementations, one for the input value and one for the output value. A 2-D table has three implementations, two for the inputs and one for the output value.

To define the implementation of tables:

- Open the implementation editor for a 1-D or 2-D table.



Implementation for: Tab_2D   Project: Module_copyimpl_DEFAULT...

Use Implementation Type: [ ]

Implementation

Transformation

Formula	ident
Conversion	f(phys) = phys
Quantization	Calculated 1.0 Qu. Exp. 0

Master: [✓ Model] Implementation

Model		Implementation	
Type	cont	Type	int32
Min	-2147483648.0	Min	-2147483648
Max	2147483647.0	Max	2147483647
		Zero not included	[ ]

Implementation Interval Adaptation

Limit Assignments [ ]

Limit to maximum bit length [ ]   Automatic

Memory Location	Default
Cache Element	Automatic

Consistency

Source	Conflict

The "Value", "X Distribution" and "Y Distribution" (2-D table only) determine which implementation you are editing.

- Select the tab for the implementation you want to edit.
- Edit the implementation.

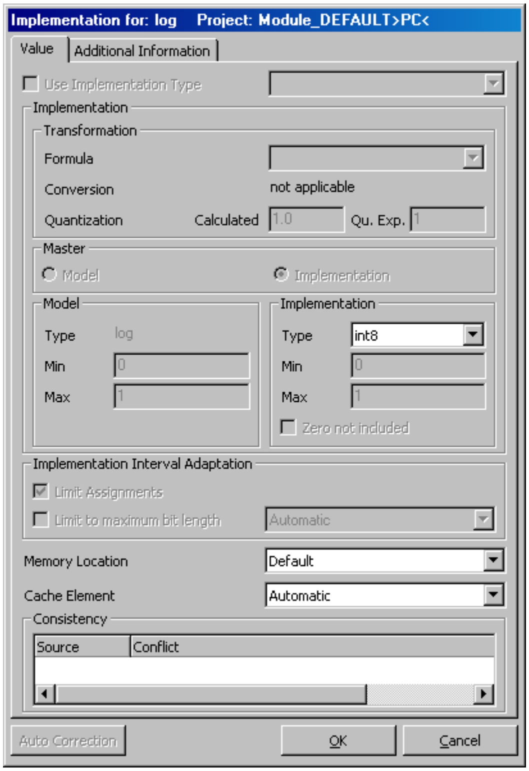
  This is like editing the implementation of a scalar element (see chapter 4.12.2 on page 479).

- Select another tab to edit the next implementation.
- Click on **OK**.

4.12.6 Implementation of Logical Elements

To specify an implementation for a logical element:

- Open the implementation editor as described in "To open the implementation editor for basic elements:" on page 480.


Implementation for: log Project: Module_DEFAULT>PC<

Value | Additional Information

☐ Use Implementation Type

Implementation
Transformation
Formula
Conversion: not applicable
Quantization: Calculated 1.0 Qu. Exp. 1

Master
○ Model ● Implementation

Model
Type: log
Min: 0
Max: 1

Implementation
Type: int8
Min: 0
Max: 1
☐ Zero not included

Implementation Interval Adaptation
☑ Limit Assignments
☐ Limit to maximum bit length: Automatic

Memory Location: Default
Cache Element: Automatic

Consistency

Source	Conflict

All fields in the "Value" tab, except the "Type", "Memory Location", and "Cache Locking" combo boxes, are disabled because they are irrelevant for the implementation of logical elements.

- In the "Type" combo box, select the implementation data type.

  The following types are available: `bit`, `int8`, `int16`, `int32`, `uint8`, `uint16` and `uint32`.
  The logical element is represented by a variable of the selected data type.

- In the "Memory Location", select the memory area where the element is located.

  The available selection depends on the current target. This setting is only relevant for experiments on microcontroller targets and is ignored in all other cases.

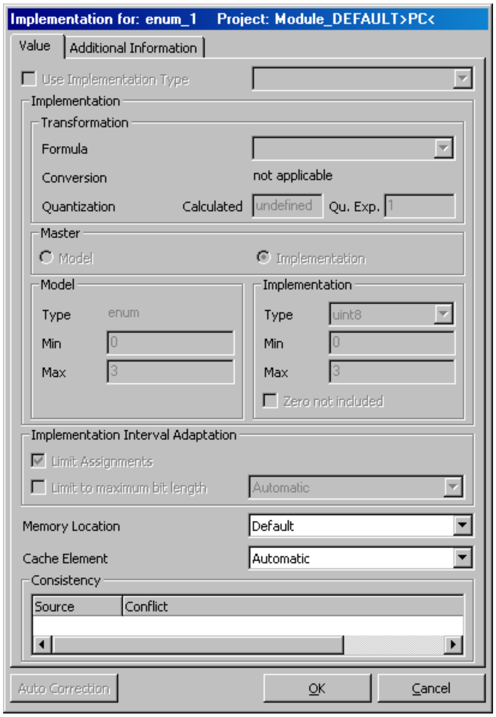
- In the "Additional Info" tab, enter enter information for your code generator.

  This information is only evaluated where applicable. The exact nature of the information you can enter here depends on your target and code generator.

4.12.7 Implementations of Enumerations

To specify an enumeration implementation:

- Open the implementation editor for the enumeration as described in "To open the implementation editor for basic elements:" on page 480.



Implementation for: enum_1   Project: Module_DEFAULT>PC<

Value | Additional Information

☐ Use Implementation Type

Implementation

Transformation

Formula

Conversion: not applicable

Quantization   Calculated: undefined   Qu. Exp.: 1

Master

○ Model   ● Implementation

Model

Type: enum

Min: 0

Max: 3

Implementation

Type: uint8

Min: 0

Max: 3

☐ Zero not included

Implementation Interval Adaptation

☑ Limit Assignments

☐ Limit to maximum bit length: Automatic

Memory Location: Default

Cache Element: Automatic

Consistency

Source	Conflict

All fields in the "Value" tab, except the "Memory Location" and "Cache Locking" combo boxes, are disabled because they are irrelevant for the implementation of logical elements.

- In the "Memory Location", select the memory area where the element is located.

  The available selection depends on the current target.
  This setting is only relevant for experiments on microcontroller targets and is ignored in all other cases.

- In the "Additional Info" tab, enter enter information for your code generator.

  This information is only evaluated where applicable. The exact nature of the information you can enter here depends on your target and code generator.

4.12.8 Method and Process Implementations

In ASCET, it is possible to have implementations not only for data sets but also for the processes or methods defined in modules and classes. With that, you can improve the overall behavior of your system.-

The implementation of methods and processes is edited from the specification editor. It cannot be edited from the project editor directly. Process and method implementations are available for both block diagram and ESDL components.

The implementation of a process (or a method) determines how the process is represented in the C code generated and where it is located during execution of the program.

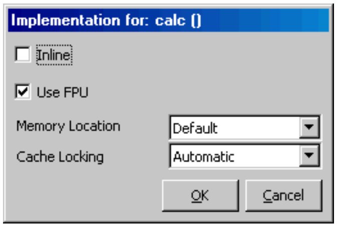
To edit a process/method implementation:

- Open the specification editor for the module or class you want to edit.
- In the "Diagrams" pane, select the process or method you want to edit.
- Select **Diagram** → **Edit Implementation**

or

- select **Edit Implementation** from the context menu.

  The implementation editor for the process/ method opens.



Implementation for: calc ()

- [ ] Inline
- [x] Use FPU

Memory Location	Default
Cache Locking	Automatic

- Activate the **Inline** option so that the method code is inserted directly into the model code.

Note

The **Inline** *option is not available for processes.*

  By this means, no function call for the method is necessary. Thus, runtime is optimized, however, at the cost of enhanced memory requirements in case the method is used multiply.

- Activate the **Use FPU** option when you want to save the Floating Point Unit registers of your microcontroller target upon task switching.

  This option is part of the ERCOS^{EK} operating system. It is effective only in connection with a microcontroller target; details are given in the ASCET-SE V5.2 User's Guide as well as the ERCOS^{EK} User's Guide.

- From the "Memory Location" combo box, select the memory area where the code should run.

  As a general rule, processes that are slow and rarely called should run in external memory whereas fast and frequently called processes should run in internal memory to improve performance.

4.12.9 Implementations of Arguments and Return Values of Methods

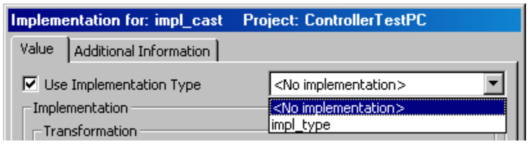
The arguments and return value of a method are implemented exactly as elements of the same type (see chapter 4.12.1, 4.12.2, 4.12.5, 4.12.6, and 4.12.7).

For scalar and logical arguments and return values, and arguments and return values of type `<enumeration>`, the "Memory location" combo box is deactivated. These elements are stored in the `STACK` memory class.

If references are used as arguments ad return values, i.e. the type `<array[*]>`, `<mat[*]>` or `<user defined>` is selected, a memory class can be selected in the "Memoy location" combo box. This memory class does *not* apply to the reference in this case, it applies to the target of the reference.

4.12.10 Implementations of Implementation Casts

Implementation casts can be implemented the same way as scalar elements (see chapter 4.12.2). The only new thing is the possibility not to implement them at all—by selecting `<No implementation>` in the "Implementation Type" combo box. `<No implementation>` is the default selection for newly created implementation casts.



Note

When `<No implementation>` *is selected for an implementation cast, it is treated as nonexistent by each code generation.*

4.12.11 Operator Implementation

In previous ASCET versions, operators in block diagrams could be implemented, too. This made it possible to improve the accuracy of operations in fixed-point arithmetic.

Note

The implementation options **Limit to maximum bit length** *and* **Zero not included** *replace operator implementations. In addition, implementation casts can be used to insert requantizations in concatenated arithmetic operations without creating additional storage space requirements.*
Therefore, no new operator implementations can be created. Existing operator implementations in older projects can be viewed, replaced by implementation casts (see "Automatic Conversion of Operator Implementations") or removed, but not edited.

Operators do not have a name in the graphic representation and do not appear in the "Elements" list. Therefore, the implementation viewer for an operator is only accessible from its icon in the graphic block.

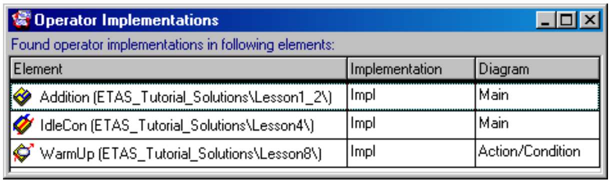
If an implementation is specified, the operator is marked in the graphic by a small line in the top left-hand corner (+). This allows the user to see where there has been an intervention.

Use the **Tools** → **Database** → **List Operator Implementations** command in the Component Manager to easily detect operator implementations.

To search for operator implementations:

- In the Component Manager, select **Tools** → **Database** → **List Operator Implementations**.

  The database is searched. Depending on its size, the search can take several seconds. The detected operator implementations are listed in the "Operator Implementations" window.



Found operator implementations in following elements:

Element	Implementation	Diagram
Addition (ETAS_Tutorial_Solutions\Lesson1_2\)	Impl	Main
IdleCon (ETAS_Tutorial_Solutions\Lesson4\)	Impl	Main
WarmUp (ETAS_Tutorial_Solutions\Lesson8\)	Impl	Action/Condition

The "Element" column lists all components/ projects containing operator implementations. The "Implementation" column lists the name of the respective implementation, and the "Diagram" column lists the diagram that contains the operator implementation.

- In the "Element" column, double-click on a component.

  The editor for the component opens. The affected operators are highlighted. You can view or delete the implementations.

To remove an individual operator implementation:

- Open the respective block diagram.
- Right-click on an operator with an implementation and select **Implementation** → **Reset** from the context menu.

  The implementation is deleted, and the line in the corner of the operator is removed.

To remove operator implementations in a component/the entire database:

- In the Component Manager, select a component from the "1 Database" field.

- Select **Component** → **Remove Operator Implementation** → **Flat** to remove the operator implementations of the component.
- Select **Component** → **Remove Remove Operator Implementation** → **Recursive** to remove the operator implementations of the component and its referenced components.

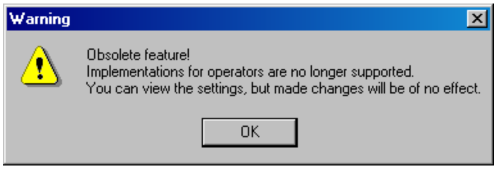
Or

- Select **Tools** → **Database** → **Convert** → **Reset Operator Implementations** to remove all operator implementations of the entire database.

To view an operator implementation:

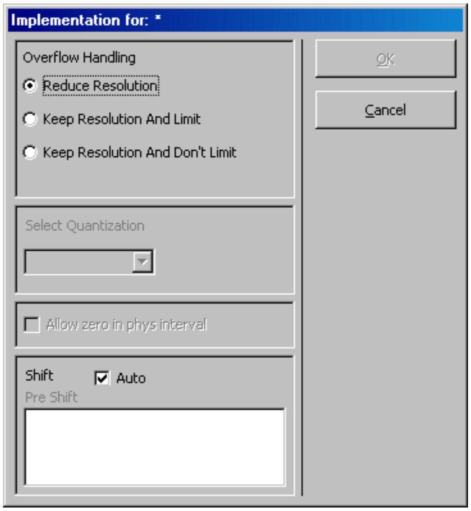
- Open the respective block diagram.
- Right-click on an operator with an implementation and select **Implementation** → **View** from the context menu.

  A warning is displayed, indicating that operator implementations are no longer supported.


Warning

Obsolete feature!
Implementations for operators are no longer supported.
You can view the settings, but made changes will be of no effect.

- Click **OK** to open the "Implementation for" window.



Implementation for: *

Overflow Handling
- [✓ Reduce Resolution]
- Keep Resolution And Limit
- Keep Resolution And Don't Limit

Select Quantization

Allow zero in phys interval

Shift: Auto ✓

Pre Shift

The window serves as a display only, you cannot change any setting.

- Click on **OK**.

The implementation for a mathematical operator determines the procedure if there is an overflow and/or the quantization (e.g. the accuracy) in the result. As with variables, this information can vary across different implementations for the relevant module. The information available for each operator is described below.

Note

The integer code generator treats the basic operations completely differently. Additional information, therefore, also depends on the specific operation.

Addition and Subtraction:

Overflow Handling:

- *Reduce Resolution* avoids overflow by shifting both inputs to the right. The shift is determined automatically.
- *Keep Resolution And Limit* does not perform a shift and uses mathematical service routines with clipping if available.

- *Keep Resolution And Don't Limit* does not perform a shift and uses normal arithmetic, thus allowing overflow to occur. This variant is used for counters which should overflow cyclically, for example.

Select Quantization:

- *Auto* selects a quantization using an optimization strategy.
- *1* selects quantization and data type of the first input.
- *2* selects quantization and data type of the second input.

Multiplication:

Overflow Handling**:**

- *Reduce Resolution* avoids overflow by shifting both inputs to the right. The shift is determined automatically.
- *Keep Resolution And Limit* does not perform a shift and uses mathematical service routines if available.
- *Keep Resolution And Don't Limit* does not perform a shift and uses normal arithmetic, thus allowing overflow to occur.

Pre-shift:

Both operands can be shifted before the operation to avoid an overflow and to re-scale each operand in a numerically meaningful way. The pre-shift can also allow full use of the value range. Integer numbers between -31 and 31 are possible for either operand; a positive number indicates left shift, negative indicates right shift. Zero means no shift at all. The following settings can occur:

- The activated *Auto* option indicates no pre-shift.
- When *Auto* is deactivated, the pre-shift is performed according to the settings in the "Pre Shift" field.
  - Line *1* determines the shift for the first operand (-31 to 31).
  - Line *2* determines the shift for the second operand (-31 to 31).

Division:

Overflow Handling**:**

- *Reduce Resolution* avoids overflow by shifting both inputs to the right. The shift is determined automatically.
- *Keep Resolution And Limit* does not perform a shift and uses mathematical service routines if available.
- *Keep Resolution And Don't Limit* does not perform a shift and uses normal arithmetic, thus allowing overflow to occur.

The **Allow zero in phys. interval** option could be activated when the generated code should not test for division by zero, even though the denominator interval includes zero. Therefore, option is an assurance to the code generator that the user himself will take care that the denominator does not assume the value zero.

Note

Wrong usage of this option can lead to severe exception errors in the control unit.

Pre-shift:

Indicates if the numerator should be maximized automatically by left shift to improve numerical accuracy. The following settings can occur:

- *Auto* left-shifts the numerator automatically.
- When *Auto* is deactivated, the pre-shift is performed according to the settings in the "Pre Shift" field.
  - Line *1* determines the shift for the first operand (-31 to 31).
  - Line *2* determines the shift for the second operand (-31 to 31).

Multiplexer, Maximum, Minimum:

Select Quantization:

- *Auto* selects a quantization using an optimization strategy.
- *1* selects quantization and data type of the first input.
- *2* selects quantization and data type of the second input.

Because these operations do not perform calculations, no overflow handling is required.

Automatic Conversion of Operator Implementations

You can delete operator implementations in older models (see page 504) or replace them automatically by the newly introduced implementation casts (see section "Implementation Casts" in the ASCET Reference Guide). Automatic replacing applies to the *entire* database and not individual components.

Rules for automatic conversion: The following conditions have to be fulfilled for an operator implementation to be converted automatically.

- The operator implementation must not contain any other quantization than **Auto** (addition, subtraction, MIN, MAX and MUX; see e.g. page 506).

Note

This is the only condition which has to be fulfilled for automatic conversion for MIN, MAX and MUX operators. The other conditions only apply to `+, -, *, /`.

- The operator output has to be connected.
- The operator output can only be connected to primitive elements.

  It cannot be connected to a component or operator or hierarchy.
- If an implementation cast is connected to the operator output, *something other* than `<No implementation>` has to be selected for this implementation cast in the combo box next to the **Use Implementation Type** option.
- The operator implementation must not contain any special pre-shift (multiplication and division; see page 507).
- If the operator is a division operator and the **Allow zero in phys. interval** option is activated in the operator implementation, the following rules also apply for the denominator input:
  – The denominator input has to be connected.
  – The denominator input can only be connected to primitive elements.

    It cannot be connected to a component or operator or hierarchy.
  – If an implementation cast is connected to the denominator input, *something other* than `<No implementation>` has to be selected for this implementation cast in the combo box next to the **Use Implementation Type** option.

If one of these conditions is not fulfilled in any implementation of the component (see chapter 4.12.1), the relevant operator has to be converted manually.

To replace an operator implementation with an implementation cast:

- Select **Tools** → **Database** → **Convert** → **Operator Implementations to Impl. Casts** in the Component Manager.

  The operator implementations of the entire database are converted into implementation casts in accordance with the above rules.

If an operator (apart from MIN, MAX, MUX) can be converted automatically, the following occurs:

- An implementation cast is created on every connection of the operator output.
- If the operator is a division operator and the **Allow zero in phys. interval** option is activated in the operator implementation, an implementation cast is created on the connection to the denominator input.
- The implementation information of the following element is accepted for every implementation cast at the output of an implemented operator.
  This is not the case for the model type; this is always `cont` for implementation casts.
- The implementation information (apart from the model type) from the previous element is accepted for implementation casts which were added at the denominator input of a division operator.

Note

For implementations of the component (see chapter 4.12.1) in which the operator has no implementation, `<No implementation>` *is selected for newly created implementation casts.*

- The overflow handling is converted accoring to the following scheme:

Implementation Cast: / Operator Implementation:	**Limit to maximum bit length**	**Reduce Resolution**	**Keep Resolution**
Reduce resolution	**X**	**X**	
Keep resolution and limit	**X**		**X**
Keep resolution and don't limit			**X**

  Each row shows the settings set for the implementation cast to replace the corresponding setting of the operator implementation.
- The operator implementation is removed.

If a MIN, MAX or MUX operator can be converted automatically, only the operator implementation is removed. No implementation cast is added.

According to the definitions above, under certain conditions implementation casts would be created with the same implementation as the element connected to their output. In these cases, no implementation cast is inserted.

If an operator cannot be converted automatically, the following occurs:

- An implementation cast is created on every connection of the operator output—even with components, operators etc. . `<No implementation>` is selected for these implementation casts in all implementations of the component.
  This implementation cast is given the relevant implementation information during manual conversion of the operator implementation.

  If this kind of implementation cast already exists on one of these connections, no other implementation cast is added to this connection.
- If the **Allow zero in phys. interval** option is activated in the operator implementation of a division operator, an implementation cast with `<No implementation>` is created on the connection of the denominator input.

  If this kind of implementation cast already exists, another one is not added.
- The operator implementation remains unchanged.

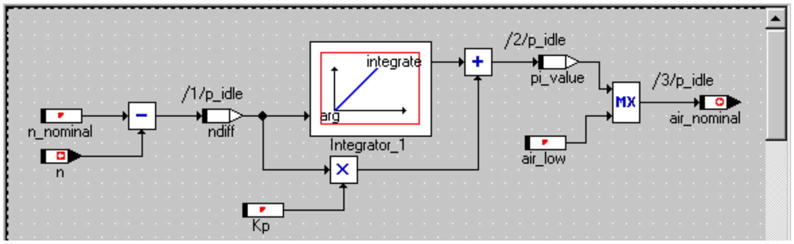
If it is not possible to convert all operator implementations automatically in the database, the following message is issued:

```
Not all operator implementations could be replaced
automatically. Please do the conversion manually.
```

Confirm this message with **OK**. The "Operator Implementations" window (see page 504) opens; it shows the components which contain the remaining operator implementations. You can now convert these manually.

4.13 Editing the Layout of a Component

When you include a component, it appears as a graphical block in the drawing area of the block diagram or project editor, with the interface elements of the component represented by the input and output pins of the block.



You can change the appearance of the graphical block with the layout editor. It is possible to add an icon, move the input and output pins around and to change the size and color of the block. Depending on the type of component, you can also enable or disable component methods or processes to determine its public interface.

The layout editor can be called from all component editors and the Component Manager.

To open the layout editor:

- Open the specification editor for the component with the layout you want to change.
- Select **Component** → **Edit Layout**.

Or

- In the browser view, select the "Layout" tab.
- Double-click the graphical block.

  The layout editor opens for the component you selected.

If you want to include a component in another component or project, you can determine whether the layout of the included component can be edited in the block diagram editor or project editor (*flexible layout*). Globally, you enable flexible layout via the **Activate flexible layout** option in the ASCET options window ("Options for Block Diagrams" on page 54). Indepentent of this option, you can enable or disable flexible layout for each class or module locally. By default, flexible layout is disabled globally and locally.

4.13.1 Editing a Class Layout

In the layout editor, you can change the appearance of the block, as well as enable and disable individual methods. If a method is disabled in the layout editor, it cannot be used when the class is referenced by another component.

To modify the diagram block:

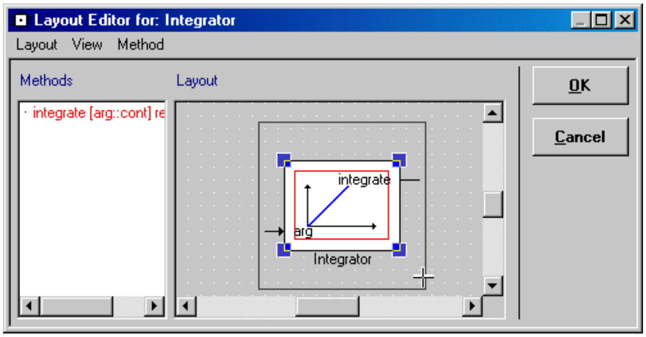
- In the layout editor, click on the block in the drawing area.

  When you are editing a layout that includes a symbol, you may easily select the symbol instead of the block—particularly if both are of similar size. The symbol, however, cannot

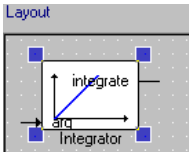
be edited in the layout editor.
In such a case, click on the *border* of the block.

The block is selected; it displays resize handles on all its corners.

- Drag the handles to resize the block.



When the block becomes *smaller* than the symbol, you can no longer resize the former in the layout editor because you will always select the symbol.



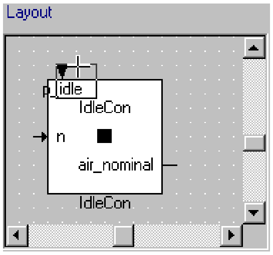
To change the block size, you have to insert the component into a block diagram, place it in the drawing area, change the layout according to section "Layout of Included Components" on page 228, and define the changed layout as default, according to page 233.

To move the ports:

The default location for inputs is on the left side of the block, the default location for the outputs is on the right side. The default location for methods is on the top side. You can move all ports.

- Drag the inputs and outputs along the sides of the block to the positions you want.

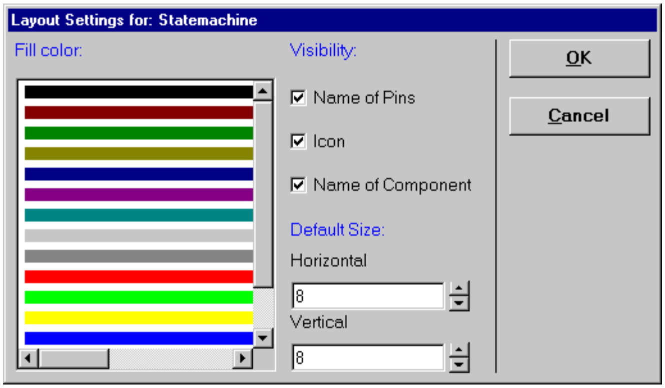
- Drag the methods along the sides of the block to the positions you want.



You cannot place one port at a position already occupied by another port.

To modify the block attributes:

- In the layout editor, select **Layout** → **Modify Attributes** to open the "Layout Settings" dialog.


Layout Settings for: Statemachine

Fill color:

Visibility:

☑ Name of Pins

☑ Icon

☑ Name of Component

Default Size:

Horizontal: 8

Vertical: 8

- Select the fill color from the "Fill color" box.
- Adjust the block size by selecting values from the "Horizontal" and "Vertical" boxes. This has the same effect as dragging the sizing handles.
- Adjust the visibility options by checking or unchecking the boxes:

  - **Name of Pins**: The input and output pins are labeled with the interface element names.
  - **Icon**: The icon referenced by the component is shown in the middle of the block.
  - **Name of Component**: The name assigned to the component in the Component Manager. The instance name, i.e. the name assigned to an individual instance of a component when it is referenced by another one, can be made visible or invisible on the drawing area.
- Click **OK** to modify the attributes for the current block.

It is possible to modify the default attributes that apply to all new blocks in a database.

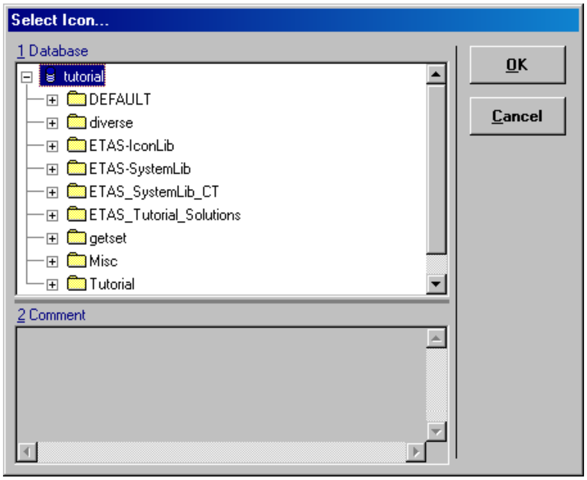
To change the default attributes for new blocks:

- Select **Layout** → **Edit Default Attributes**. This brings up the "Layout Settings" dialog box again. You can enter the same settings as described above, but they will not affect the current block. The settings define the default attributes that are applied to all newly created components.
- Select **Layout** → **Set Attributes to Default** to apply the default attributes to the current graphical block.

You can use an icon in a component which will then be shown in the graphical block of that component. ASCET icons are stored in the database and can be accessed via the Component Manager.

To assign an icon to a class:

- In the layout editor, select **Layout** → **Select Icon** to open the "Select Icon" dialog.



- From the "Items" list select the icon you want to assign to the block.
- Click **OK** to assign the icon.

  The icon is assigned to the component in the layout editor.

Creating and editing symbols is described in chapter "The Icon Editor" on page 558.

The layout editor offers severel possibilities to set up the dispaly.

To modify the layout editor display:

- Select **View** → **Redraw** to modify the display.
- Select **View** → **Grid** to open the "Grid Option".

  This dialog box is described in section "Viewing and Printing Diagrams" on page 263.
- Select **View** → **Print** to print the layout.
- Click **OK** to exit the layout editor.

You can modify the public interface by enabling or disabling public access to methods and variables.

To enable and disable methods:

- In the "Methods" list of the layout editor, select the method you want to enable or disable.
- Select **Method** → **Disable** to disable public access to the selected method.
- Select **Method** → **Disable** or **Enable** to enable public access to the selected method.

  By default, all the methods are enabled. They are displayed in red in the "Methods" list of the Layout Editor. A disabled method is not displayed in the layout pane and is displayed in black in the "methods" list.

4.13.2 Editing the Layout of Other Components

- *Modules:* Editing the layout of a module is the same as for classes, except that processes instead of methods and messages instead of elements can be enabled and disabled.
- *Continuous time blocks.* Methods and elements cannot be enabled or disabled in the layout editor. The appearance of the block is modified in the same way as that of a class.
- *State machines* and *conditional tables:* The same as for classes.
- *Boolean Tables:* Same as for classes, except that methods cannot be disabled.

4.14 Arithmetic Services

ASCET offers its users the ability to define so-called *arithmetic services* and use them to optimize elementary operations, such as addition operations, and to extend such operations by special properties, such as limiters or overflow handling.

An arithmetic service is a C function, which is made available by the user and can be called, in place of a standard operation, from within the code generated by ASCET. For example, in place of the normal add operation,

```
c = a + b;
```

a function call is generated:

```
c = add(a, b);
```

The code generator can replace, among others, the following standard operations with user-defined service function calls:

- addition, subtraction, multiplication and division
- negation and absolute value
- interpolation point search and interpolation
- combined operations, such as multiplication with subsequent division

Arithmetic services can be used for all physical specification types (ESDL, block diagrams) and all complex elements (classes, modules, finite automata or state machines), and are supported for all target types (micro-controller targets, experimental targets [rapid prototyping] and PC) both in off-line and in online experiments with one of the implementation code experiments (see page 412).

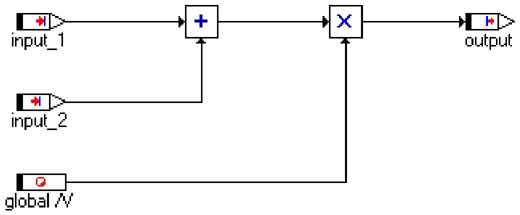
ASCET finds the information on the presence and type of services from a `services.ini` file, which is stored for each target in a corresponding target directory (`Ascet5.2\target\<target>`). These files match the Windows standard for `*.ini` files; they can be created and edited using any common text editor or using the AS Editor provided (see chapter 4.14.5).

The arithmetic services are not made available by ASCET directly. The user has the option of making the service functions available in a form best suited for the pertinent application of them: as a library, object code, C code or with the aid of the ASCET C code editors.

4.14.1 The Functionality of an Arithmetic Service

In addition to C code descriptions of classes, modules etc., ASCET offers the ability to describe these graphically (in block diagrams) or in an abstract language (ESDL). In order to generate executable code for a control unit or an experiment from such descriptions, the descriptions have to be converted into

C code. For this purpose, ASCET is equipped with a code generator. The following example is provided to illustrate why the code generator needs an arithmetic service and what effect this has. .



The block diagram above graphically depicts two operations: one addition and one multiplication. There are two possible ways of converting this function into C code.

1. using standard C operations, + and *
2. using separate functions for the addition and multiplication operations

For the addition operation, the standard operation would look like this:

```
output := input_1 + input_2;
```

The following would be feasible as a second option:

```
uint8 add(uint8 summand_1, uint8 summand_2)
{
   return summand_1 + summand_2
}
...
output := add(input_1, input_2);
```

By default, ASCET would apply the standard + operator for the simple addition. By creating an arithmetic service, the user can specify that the second variant is to be applied.

This requires to steps:

1. First, prepare the function code for the `uint8 add(uint8 summand_1, uint8 summand_2)` function.

   This is typically done outside of ASCET, for example in the form of a function library, but can also be carried out within the model.
2. Then define the service call to be applied.

For the simple addition in the example above, this would appear as such:

```
+|u8|u8|u8 = add(%i1%, %i2%)
```

The syntax of this definition is explained in the sections below.

Based on this definition, the code generator generates the `add(%i1%, %i2%)` function call in place of the standard `"+"` operation for each addition of two `uint8` values that yield one `uint8` result. In doing so, the formal arguments `%i1%` and `%i2%` are replaced by the corresponding, concrete arguments (e.g. `input_1` and `input_2`).

4.14.2 Defining Arithmetic Services

Arithmetic services consist of two parts: the definition of the service call and the preparation of the function code. While the function code does not have to follow any hard rules except that it must be valid C code, which has to be provided to ASCET (see chapter 4.3.3), the definition of these service calls must be of a form that can be understood by ASCET. The definition of the service call for an arithmetic service follows a strict syntax:

```
operation|opType1|opType2|opType3|resType=function
```

Every definition consists of two parts:

- Function key
  - `operation|opType1|opType2|opType3|resType`
- Function call in C syntax
  - `function`

The function key serves to unambiguously assign a standard operation to an arithmetic service. The function call exactly represents the function call to be used by the code generator.

Every theoretically possible standard operation is to be described by exactly one key, and there can be only one defined service per key.

Function Key

Function keys are always formed according to the following syntax:

```
operation|opType1|opType2|opType3|resType
```

The individual elements of the key carry the following meanings:

- `operation` - a character string that unambiguously identifies the corresponding operation;
- `opType` - a character string that unambiguously describes the type of operands necessary;
- `resType` - a character string that unambiguously describes the type of result of the operation;

For all calculations that occur in ASCET, there is a corresponding arithmetic service, which the user can customize and optimize to suit specific needs. The table below lists all of the arithmetic services that are understood by ASCET, as well as the corresponding number and the type of the parameters (O = operation; 1,2,3 = operand1, 2, 3; R = result) and the meaning of the function. In addition, it specifies a standard function definition as can be generated automatically by the AS Editor (see chapter 4.14.5).

Function Key	**Standard Function Definition**	**Parameters**	**Meaning**
`abs|*|*`	`abs_%t1%_%tr% (%i1%)`	`O|1|R`	Deriving the absolute value
`neg|*|*`	`neg_%t1%_%tr% (%i1%)`	`O|1|R`	Negating a value
`+|*|*|*`	`add_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Addition of two values
`+l|*|*|*`	`addl_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Addition of two values with saturation[a]
`-|*|*|*`	`sub_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Subtraction of two values
`-l|*|*|*`	`subl_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Subtraction of values with saturation[a]
`*|*|*|*`	`mul_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Multiplication of two values
`*l|*|*|*`	`mull_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Multiplication of values with saturation[a]
`/|*|*|*`	`div_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Division of two values
`/l|*|*|*`	`divl_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Division of two values with saturation[a]
`%|*|*|*`	`mod_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Modulo calculation of two values
`%l|*|*|*`	`modl_%t1%%t2%_%tr% (%i1%, %i2%)`	`O|1|2|R`	Modulo calculation of two values with saturation[a]
`*>|*|*|*|*`	`mul_r_%t1%%t2%_%tr% (%i1%, %i2%, %i3%)`	`O|1|2|3|R`	Multiplication of two values with subsequent right shift

a: Saturation refers to the act of limiting results to within maximum possible range of values for the relevant type of result.

Function Key	Standard Function Definition	Parameters	Meaning
`*>l|*|*|*|*`	`mul_rl_%t1%%t2%_%tr% (%i1%, %i2%, %i3%)`	`O|1|2|3|R`	Multiplication of two values with subsequent right shift and result limitation
`/>|*|*|*|*`	`div_r_%t1%%t2%_%tr% (%i1%, %i2%, %i3%)`	`O|1|2|3|R`	Division of two values with subsequent right shift
`/>l|*|*|*|*`	`div_rl_%t1%%t2%_%tr% (%i1%, %i2%, %i3%)`	`O|1|2|3|R`	Division of two values with subsequent right shift and saturation[a]
`*/|*|*|*|*`	`muldiv_%t1%%t2%%t3%_%tr% (%i1%, %i2%, %i3%)`	`O|1|2|3|R`	Multiplication of two values with subsequent division
`*/l|*|*|*|*`	`muldivl_%t1%%t2%%t3%_%tr%(%i1%, %i2%, %i3%)`	`O|1|2|3|R`	Multiplication of two values with subsequent division and saturation[a]
`getAt1|*|*`	`CharTable1_getAt_%t1%_%tr%(%ct%, %x%)`	`O|1|R`	Characteristic access
`getAt1R|*|*`	`CharTable1_getAtR_%t1%_%tr%(%ct%, %x%)`	`O|1|R`	Rounded characteristic access
`getAt2|*|*|*`	`CharTable2_getAt_%t1%&t2%_%tr%(%ct%, %x%, %y%)`	`O|1|2|R`	Characteristic diagram access
`getAt2R|*|*|*`	`CharTable2_getAtR_%t1%&t2%_%tr%(%ct%, %x%, %y%)`	`O|1|2|R`	Rounded characteristic diagram access
`getAtFixed1|*|*`	`CharTable1_getAtFixed_%t1%_%tr%(%ct%, %x%)`	`O|1|R`	Access to fixed charac-teristic curve
`getAtFixed1R|*|*`	`CharTable1_getAtFixedR_%t1%_%tr%(%ct%, %x%)`	`O|1|R`	Rounded access to fixed characteristic curve
`getAtFixed2|*|*|*`	`CharTable2_getAtFixed_%t1%&t2%_%tr%(%ct%, %x%, %y%)`	`O|1|2|R`	Access to fixed charac-teristic map
`getAtFixed2R|*|*|*`	`CharTable2_getAtFixedR_%t1%&t2%_%tr%(%ct%, %x%, %y%)`	`O|1|R`	Rounded access to fixed characteristic map
`interpolGroup1|*|*`	`GroupTable1_getAt_%t1%_%tr%(%ct%, %x%)`	`O|1|R`	Interpolation routine for group characteristic curve

a: Saturation refers to the act of limiting results to within maximum possible range of values for the relevant type of result.

Function Key	Standard Function Definition	Parameters	Meaning
`interpolGroup1R|*|*`	`GroupTable1_getAtR_%t1%_%tr%(%ct%, %x%)`	`O|1|2|R`	Interpolation routine (rounded) for group characteristic curve
`interpolGroup2|*|*|*`	`GroupTable2_getAt_%t1%&t2%_%tr%(%ct%, %x%, %y%)`	`O|1|2|R`	Interpolation routine for group characteristic map
`interpolGroup2R|*|*|*`	`GroupTable2_getAtR_%t1%&t2%_%tr%(%ct%, %x%, %y%)`	`O|1|2|R`	Interpolation routine (rounded) for group characteristic map
`searchDistrib|*|*`	`Distribution_search_%t1%_(%i1%)`	`O|1`	Distribution search
`getHighPart`	`_(%i1%)`	`O`	Returns the highest bit of a value

Tab. 4-1 Arithmetic Services Supported by ASCET

Note

The number and order of the parameters in Tab. 4-1 is the default. In principle, kind, number and order of the parameters is not restricted (as long as the C syntaxt is correct); you can add more parameters for special purposes (cf. "Specifying an entry:" on page 549).

Allowable Types

Character String	Type
`u8`	unsigned integer 8 bit
`u16`	unsigned integer 16 bit
`u32`	unsigned integer 32 bit
`s8`	signed integer 8 bit
`s16`	signed integer 16 bit
`s32`	signed integer 32 bit
`*`	All of the types mentioned above are allowed

Tab. 4-2 Allowable Types for Operands and Results

The presence of a result type and the number of operands depend on the selected operation. For example, an addition has two operands, while a negation has only one, and a multiplication with right shift requires three.

Function Declaration

The function declaration describes in C syntax the function call that is to be applied by the code generator in place of the standard operation. If, for example, a function is defined in C as

```
int Add(int operand1, int operand2){...},
```

then the corresponding function call would be

```
Add(operand1, operand2);
```

Because the concrete character strings for `operand1` and `operand2` are not known for the declaration of the function in the `services.ini` file, formal arguments are used in place of these. These placeholders are recognized by the code generator, which replaces them by corresponding character strings during the code generation process. The function call above would appear as follows in the `services.ini` file:

```
Add(%i1%, %i2%);
```

Tab. 4-3, Tab. 4-4, and Tab. 4-5 provide a list of all arguments supported by the code generator.

Argument	**Meaning**
`%i1%`	Expression for the 1st operand
`%i2%`	Expression for the 2nd operand
`%i3%`	Expression for the 3rd operand
`%t1%`	Short type (e.g. `u8`) for the 1st operand
`%t2%`	Short type (e.g. `u8`) for the 2nd operand
`%t3%`	Short type (e.g. `u8`) for the 3rd operand
`%ft1%`	Long type (e.g. `uint8`) for the 1st operand
`%ft2%`	Long type (e.g. `uint8`) for the 2nd operand
`%ft3%`	Long type (e.g. `uint8`) for the 3rd operand
`%tr%`	Short type (e.g. `u8`) for the result
`%ftr%`	Long type (e.g. `uint8`) for the result
`%c1%`	Automatically determined type-cast of the first operand
`%c2%`	Automatically determined type-cast of the second operand

Argument	Meaning
`%c3%`	Automatically determined type-cast of the third operand
`%op%`	Name of the operation
`%il%`	Bit length of input

Tab. 4-3 Formal arguments for the `Mul` und `Div` (with and without shift and limitation), `Add`, `Sub`, `Muldiv` (with and without limitation), `Mod`, `Neg`, `Abs`, `getHighPart` operations.

Argument	Meaning
`%ct%`	Expression for characteristic curve or map
`%x%`	Expression for x interpolation point
`%y%`	Expression for y interpolation point (characteristic map only)
`%tx%`	Short type (e.g. `u8`) for the x value
`%ty%`	Short type (e.g. `u8`) for the y value (characteristic map only)
`%tv%`	Short type (e.g. `u8`) for the result
`%ftx%`	Long type (z.B. `uint8`) for the x value (fixed characteristic curve/ map only)
`%fty%`	Long type (z.B. `uint8`) for the y value (fixed characteristic curve/ map only)
`%ftv%`	Long type (e.g. `uint8`) for the result (fixed characteristic curve/ map only)
`%fmst%`	Long type, determined by the maximum size (maxSize) of the table (fixed characteristic curve/ map only)
`%xd%`	Expression for x distribution (group characteristic curve/map only)
`%yd%`	Expression for y distribution (group characteristic map only)
`%xlen%`	Number of x distribution points
`%ylen%`	Number of y distribution points (characteristic map only)
`%xdind%`	Index of lower sample point (x dimension; distribution and group characteristic curve/map only)
`%xdoff%`	Expression for offset between lower and upper sample point (x dimension; distribution and group characteristic curve/map only)
`%xddis%`	Expression for distance between lower sample point and interpolation point (x dimension; distribution and group characteristic curve/ map only)

Argument	Meaning
`%ydind%`	Index of lower sample point (y dimension; group characteristic map only)
`%ydoff%`	Expression for offset between lower and upper sample point (y dimension; group characteristic map only)
`%yddis%`	Expression for distance between lower sample point and interpolation point (y dimension; not fixed characteristic map)

Tab. 4-4 Formal arguments for the `Getat1`, `Getat2`, `Getatfixed1`, `Getatfixed2`, `Interpolgroup1`, `Interpolgroup2` operations.

Argument	Meaning
`%i1%`	Expression for the input value of a distribution
`%t1%`	Short type for the distribution input value
`%ft1%`	Long type for the distribution input value
`%c1%`	Automatically determined type-cast of the input value of a distribution
`%d%`	Distribution
`%xdind%`	Index of a value within an x distribution
`%xdoff%`	Offset of a value within an x distribution
`%xddis%`	Distance between two values within an x distribution

Tab. 4-5 Formal arguments for the `Searchdistrib` operation.

The multitude of possible arguments is based on the syntax of possible C function calls:

```
(%ftr%)functionname(%i1%, %i2%, %i3%);
```

Additionally, the code generator can generate the concrete function names as long as these follow this syntax:

```
%op%_%t1%%t2%%t3%_%tr%
```

Function names can be selected in any manner, though we recommend using the syntax described above.

4.14.3 Creating and Saving Arithmetic Services

The definitions of arithmetic services are maintained outside of ASCET in a `services.ini` file. A file of this type is stored separately for each target in a corresponding target directory. The file complies with the Windows standard for `*.ini` files.

To be able to quickly change the ASCET code generator for different requirements within a target, and thus gain the ability to use self-defined arithmetic service routines with great flexibility, it is possible to define and keep various sets of arithmetic services within one `services.ini` file. When it initializes, ASCET loads all of the information from the file for the current target, thus making it possible to apply any of the defined sets each time the code generator is run.

The entries in these files must follow this syntax (in BNF):

```
SERVICE-FILE ::= [ENTRY]+

ENTRY ::= [FUNCTION] | [COMMENT] | [SET]

COMMENT ::= ";" Σ*

SET ::= "["Σ+"]"

FUNCTION ::= [OPERATION]("|"[OPERAND])+ "=" Σ*

OPERATION ::= "abs" | "neg" | "+" | "-" | "*" | "/"
   | "%" | "+l" | "-l" | "*l" | "/l" | "*>" | "/>"
   | "*>l" | "/>l" | "*/" | "*/l" | "getAt1"
   | "getAt1R" | "getAt2" | "getAt2R" | "getAtFixed1"
   | "getAtFixed1R" | "getAtFixed2" | "getAtFixed2R"
   | "interpolGroup" | "interpolGroup1"
   | "interpolGroup1R" | "interpolGroup2"
   | "interpolGroup2R" | "searchDistrib"
   | "searchDistribR" | "getHighPart"

OPERAND ::= "*" | "u8" | "u16" | "u32" | "s8" | "s16"
   | "s32"
```

For more details regarding the `services.ini` files and how to create and edit them, see chapter 4.14.5 "The Interface Editor for Arithmetic Services".

4.14.4 Using Arithmetic Services in ASCET

ASCET supports the use of arithmetic services:

- for physical specifications in ESDL and in the block diagram;
- in classes, modules and state machines;
- for off-line and online experiments;
- for micro-controller targets, experimental targets (rapid prototyping) and PC.

The use of arithmetic services is directly linked to the code generation. The code generator takes the information regarding the function to be selected from the data types of the elements, which are linked with the corresponding inputs and outputs of an operation.

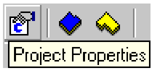
If, for example, an addition has two inputs of the "unsigned integer 8 bit" type (`uint8`) and one output of the "unsigned integer 16 bit" type (`uint16`), the code generator searches the current set for the `+|u8|u8|u16` key. If the key is found, the code generator uses the function call stored there; otherwise, depending on the type of the operation, it either uses the standard operation or generates an error message.

Selecting a Set

Selecting a set of arithmetic services takes place within the scope of a projects in ASCET.

To select a set of arithmetic services:

- Open the project to whose scope the arithmetic services you want to use apply.
- In the project editor, click the **Project Properties** button.



  The "Project Properties" dialog window opens in the "Build" node.
- In the "Code Generator" combo box, select the `Implementation Experiment` or `Object Based Controller` entry.

Note

*The settings for the "Fixed Point Codegeneration" are **not** available for* `Physical Experiment` or `Quantized Physical Experiment`.

- Open the **Integer Arithmetic** node.

  All available sets of arithmetic services are listed in the "Arithmetic Service set" combo box.
- Select the set you want to use from the combo box.

Note

When a new project is created, the first set found in the current `services.ini` *file is applied automatically for the project. If the current file does not contain any sets, or contains only an empty set, or the first set in the file is an empty set labeled* `[None]`, `<None>` *is preselected automatically in the "Arithmetic Service set" combo box.*

You can use a different set (if available) each time you run the code generator. You can also choose not to use a set (by selecting `<None>`). In this case, the code generator applies the standard operations.

ASCET saves the information regarding the set that was last selected and last used for each individual project. If a set that is not stored in the current `services.ini` file is configured for a project (for example if a project has been loaded from another computer, which uses another file), the following error message is generated:

```
The selected set of arithmetic services <...> is not
available. Please check the services.ini file.
```

If a set does not yet have functions defined in it, which are required by the code generator to generate code, either an error message is displayed, or the standard operation is applied and a warning displayed – depending on the type of operation. For more information on this, refer to the section "Potential Error Conditions".

A special case for using arithmetic services arises when constants are used in an operation. Constants defined in ASCET and literals do not have any specific type. If the user has selected to use arithmetic services, types are automatically assigned to these values during the code generation process. In this case, the code generator selects the first type from the list that is suitable for the value of the constant:

```
sint8, uint8, sint16, uint16, sint32, uint32
```

An exception to this rule occurs if a binary operator has as its input both a constant of an undefined type and a variable with a defined type.

- If the value of the constant is positive and the type of the variable is `unsigned`, the constant is assigned the same type as the variable. The purpose of this exception is to achieve, in as many cases as possible, a uniform type for the inputs of an operation.
- If the value of the constant is negative, a signed type must be selected for it. There are two possibilities:
  - The variable is `unsigned`: In this case, type uniformity is impossible.
  - The variable is `signed`: If the value of the constant conforms to the type of the variable, the type of the variable is assigned to the constant. Otherwise, type uniformity is impossible.

Potential Error Conditions

If the use of arithmetic services is enabled, the code generator depends on the validity and completeness of the arithmetic services.

The following error message is generated for all targets if an arithmetic service is required for an operation but is not found in the selected set:

```
Arithmetic service <name> required but not defined
```

This behavior is limited to all operations that are derived from purely arithmetic operations (+, -, *, /, abs, neg).

Note

*Unlike earlier versions of ASCET, **no** standard operations are used in this case by this version!*

To ensure that a corresponding arithmetic service is defined for each operation, the standard operations can be integrated into the corresponding set by adding wildcard functions. In this case, a corresponding entry must be present for each operation that is derived from an arithmetic operation (`+`, `-`, `*`, `/`, `abs`, `neg`).

For an addition, this entry would appear as follows:

```
+|*|*|*=(%i1% + %i2%)
```

For a combined multiplication/division, it would appear as such:

```
*/|*|*|*|*=((%i1% * %i2%) / %i3%)
```

The code generator then replaces the wildcards `*`) with all theoretically possible combinations of types.

Note

If, for a special type combination (such as `+|u8|s8|s8 = add_u8s8_s8(%i1%, %i2%)`*), there is a function definition in the current set, it takes precedence over the standard operation.*

ASCET generates this warning

```
Usage of arithmetic service <name> requires extra
code for requantization
```

if all elements that are directly connected to an arithmetic operation (`+`, `-`, `*`, `/`, `abs`, `neg`) are implementation casts and the user has specified differing quantizations for these.

Note

If there are no defined functions in the current set for the services, `getAt..`*,* `searchDistrib`*,* `interpolGroup..`*,* `getHighPart` *and* `modulo`*, the code generator uses the standard operations from ASCET **without** generating a warning message or notification.*

4.14.5 The Interface Editor for Arithmetic Services

The arithmetic services interface editor (henceforth AS editor) is included with ASCET as a tool which can be used for creating and editing files that contain interface definitions for arithmetic services. These files are named `services.ini`, and they have to be located in each ASCET target directory.

The files for arithmetic services (AS files) contain the interface definitions that are necessary for generating code with ASCET. The structure of AS files conforms to the Windows standard for `*.ini` files. These files contain only definitions of sets, interface definitions and comments.

A set definition is a character string enclosed in square brackets ([ ]), an interface definition takes the form of any character string that conforms to predetermined syntax, and a comment is any character string that starts with a semicolon.

An AS file should contain at least one set and no more than 255 sets of arithmetic services. A set begins with the declaration of its name and contains all of the interface definitions below it up to the next set definition or the end of the file.

An AS file might look like this:

```
[Set name]
Function
Function
...
[Set name]
;Comment
Function
Function
...
```

See chapter 4.14.3 for a detailed description of the syntax of the AS file layout.

The Functions of the AS editor

The AS editor can open existing AS files, create new AS files and save changes made to these files.

Furthermore, the AS editor provides these functions:

- creating a new (empty) set
- deleting a set
- duplicating a set

- renaming a set
- creating a new (empty) interface definition
- deleting an interface definition
- modifying an interface definition in all of its parts
- creating a standard function call for an interface definition
- verifying the syntax of an entry
- converting an interface definition to a comment
- converting a comment to an interface definition
- finding interface definitions within a set

All changes made to an interface definition are done so by selecting various, predefined values from selection lists. Changes to an AS file cannot be entered manually in the AS editor.

The AS editor only allows the user to create entries that are syntactically correct, and only creates AS files that conform completely to the requirements of the ASCET code generator.

Launching the AS Editor

The AS editor is a separate tool, which can be launched independently of or from within ASCET.

Launching the AS editor from ASCET:

- In the component manager, open the **Tools** → **Arithmetic Service Editor** menu.

  This menu contains all of the targets that are installed on your computer. The **>PC<** entry is always available.

- Select the target whose AS file you want to edit, such as **Tools** → **Arithmetic Service Editor** → **>PC<**.

  The AS editor starts and opens the AS file for the target you have selected.

  If there is no `services.ini` file in the target directory, the AS editor automatically creates a new file and names it accordingly.

Launching the AS editor from Windows Explorer:

You can launch the AS Editor independently of ASCET from the Windows Explorer.

- Open Windows Explorer.
- Switch to the `ETAS\Ascet5.2\ASEditor` subdirectory of your ASCET installation.
- From here, run the `Editor.exe` application.

  The AS editor starts.

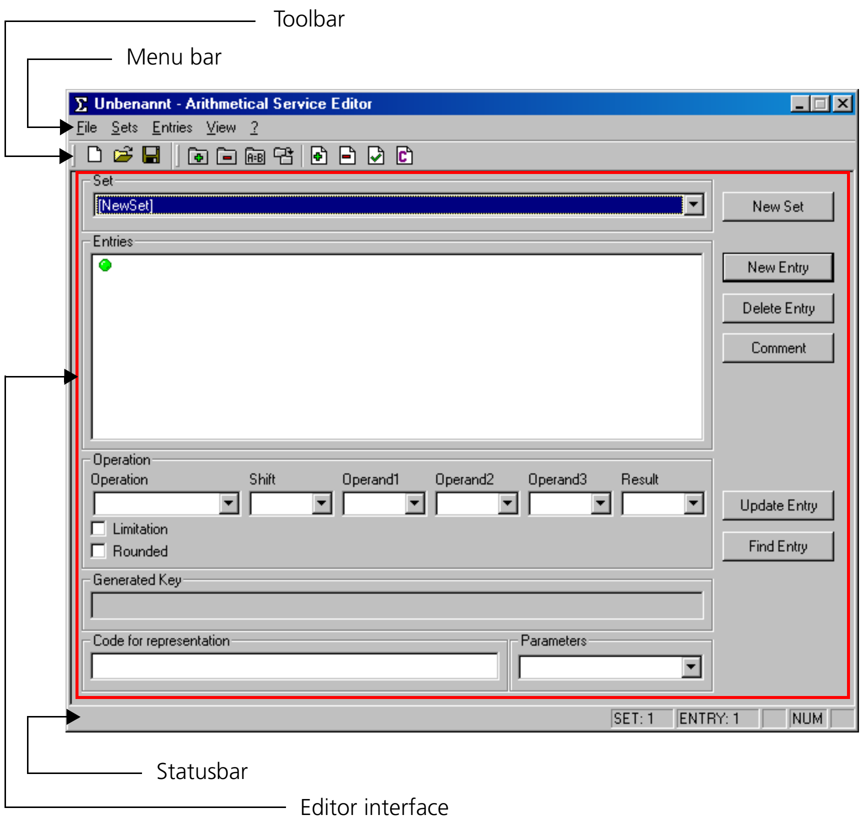
Launching the AS editor from the ASCET Start menu:

As a third option, you can launch the AS editor from the ASCET Start menu.

- In the Windows taskbar, click **Start**.
- From the **ETAS** program group, select the **ASCET5.2** → **AS-Editor** entry.

  The AS editor starts.

After you start the AS editor, the program window opens. The following screenshot provides an overview of its most important components.



Descriptions of the Menu Commands:

The commands provided on the menu bar allow you to call up all file functions (**File**), edit sets (**Sets**) and entries (**Entries**), change the appearance of the AS Editor (**View**) and call up information about the program (**?**).

- **File**
  - *New (<CTRL> + <N>)*

    Creates a new (empty) AS file.

  - *Open (<*CTRL*> + <*O*>)*

    Opens an existing AS file.
  - *Save (<*CTRL*> + <*S*>)*

    Saves the current AS file.
  - *Save as*

    Saves the current AS file under a new name.
  - *1 .. 4*

    Opens one of the 4 AS files that have been opened most recently.
  - *Exit*

    Closes the AS editor.
- **Sets**
  - *New*

    Creates a new (empty) set.
  - *Copy*

    Creates a copy of the current set.
  - *Rename*

    Renames the current set.
  - *Delete*

    Deletes the current set.
- **Entries**
  - *New*

    Creates a new (empty) entry.
  - *Update*

    Applies the changes for the current entry.
  - *Delete*

    Deletes the current entry.
  - *Comment*

    Converts the current entry to a comment.
  - *Find*

    Searches for an entry.
- **View**

– *Symbolbar*

  Hides/shows the standard toolbar.

– *Statusbar*

  Hides/shows the statusbar.

– *Toolbar*

  Hides/shows the specific toolbar.

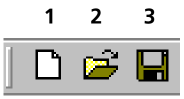
- **?**

  – *About Editor*

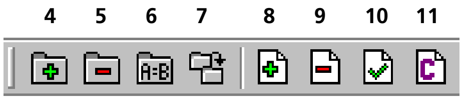
    Displays information about the program.

Descriptions of the Toolbars:

The AS editor provides two toolbars. The first is a standard Windows toolbar, and the other contains commands specific to the AS Editor.



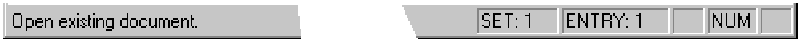
1. Creates a new (empty) AS file.
2. Opens an existing AS file.
3. Saves the current AS file.



4. Creates a new (empty) set.
5. Deletes the current set.
6. Renames the current set.
7. Creates a copy of the current set.
8. Creates a new (empty) entry.
9. Deletes the current entry.
10. Applies the changes for the current entry.
11. Converts the current entry to a comment.

The Statusbar:

The statusbar provides tips and information on the functions of graphic operating elements and displays the index of the current set and of the current entry. It also shows the status of the <NUM>, <CAPS LOCK> and <SCROLL LOCK> buttons.


Open existing document.
SET: 1 ENTRY: 1 NUM

Changing the view of the AS editor:

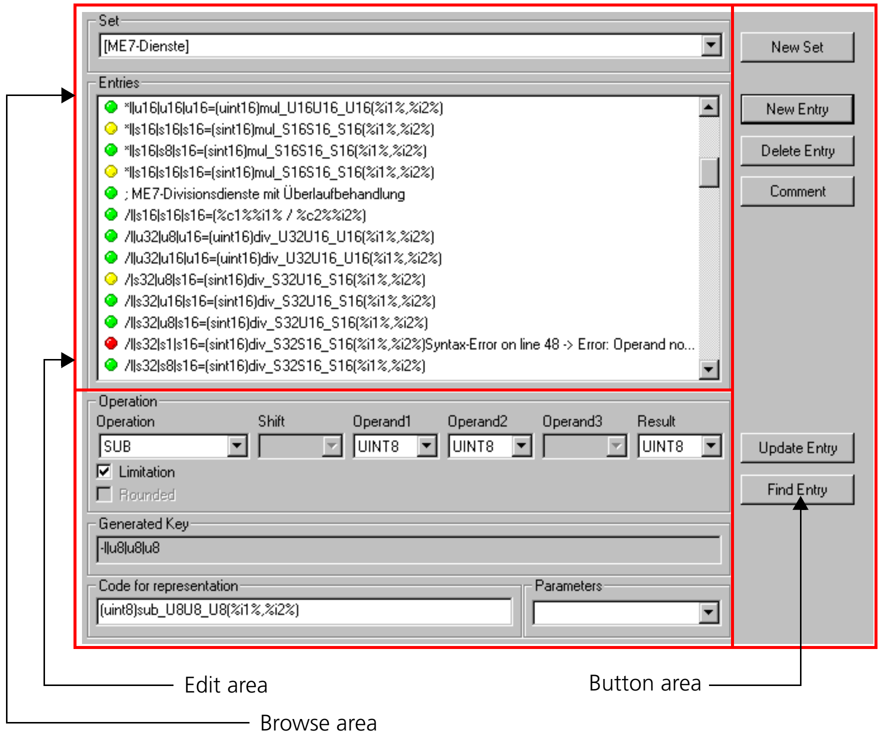
Use the **View** menu to toggle the view of various elements of the AS editors.

- Click the **View** → **Symbolbar** menu item to hide or show the standard toolbar.

  This menu item is checked when the toolbar is visible.
- Click the **View** → **Toolbar** menu item to hide or show the editor toolbar.
- Click the **View** → **Statusbar** menu item to hide or show the statusbar.

The Editor Interface

The editor interface is split into three areas.



Set: [ME7-Dienste]

Entries:

```
● *||u16|u16|u16=(uint16)mul_U16U16_U16(%i1%,%i2%)
● *||s16|s16|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● *||s16|s8|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● *||s16|s16|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● ; ME7-Divisionsdienste mit Überlaufbehandlung
● /||s16|s16|s16=(%c1%%i1% / %c2%%i2%)
● /||u32|u8|u16=(uint16)div_U32U16_U16(%i1%,%i2%)
● /||u32|u16|u16=(uint16)div_U32U16_U16(%i1%,%i2%)
● /|s32|u8|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|u16|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|u8|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|s1|s16=(sint16)div_S32S16_S16(%i1%,%i2%)Syntax-Error on line 48 -> Error: Operand no...
● /||s32|s8|s16=(sint16)div_S32S16_S16(%i1%,%i2%)
```

Operation:

Operation	Shift	Operand1	Operand2	Operand3	Result
SUB		UINT8	UINT8		UINT8

☑ Limitation
☐ Rounded

Generated Key: -||u8|u8|u8

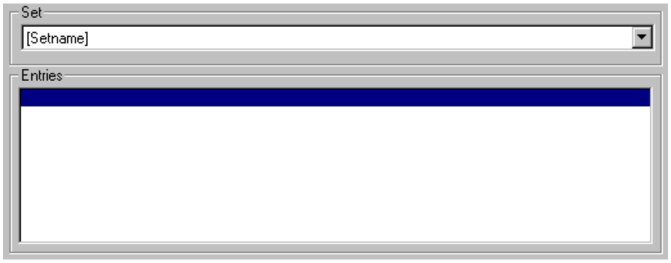
Code for representation: (uint8)sub_U8U8_U8(%i1%,%i2%)

Parameters:

The *browse area* displays all of the information and entries of an AS file. The *edit area* is where you can make changes to these entries, and you can create, update and delete the entries using the commands provided in the *button area*.

The Elements of the Browse Area:

The *browse area* displays all of the information and entries of the current AS file. After a file has been loaded successfully, all of the sets found within it are listed by name in the "Set" combo box. All of the entries belonging to the set selected in the combo box are listed in the "Entries" area. If a different set is selected in the combo box, the entries in the "Entries" field are updated accordingly.



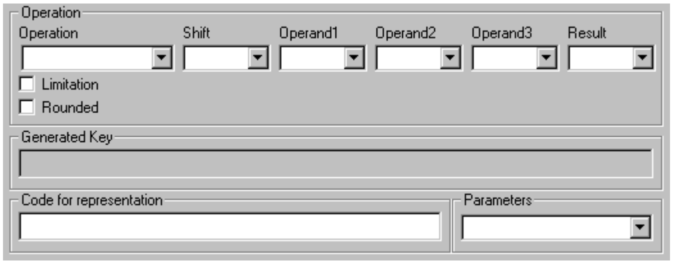
The "Set" field is a combo box, allowing you to open a list by clicking the arrow and then select exactly one element by mouse. The entries in this list cannot be edited directly and are generated by the program.

The "Entries" field is a list box. It also allows you to select exactly one element from the list. Likewise, the entries here cannot be edited directly and are entered into the field by the program.

The LED symbol in front of each entry indicates the current status of the entry. Green means that the entry is syntactically correct; red means the entry contains a syntax error. Yellow identifies entries that are syntactically correct, but do not fulfill the rules of an AS file (for example in the case of duplicate entries).

The Elements of the Edit Area:

The edit area displays the data pertaining to an entry in expanded form. Here you can also modify any of the values that form the entry.



The "Operation" field is a combo box from which you can select exactly one entry. The entries in this list show all possible and permitted operations and cannot be edited. If you select an operation from this list, any fields that are impertinent to the operation are automatically disabled. The following operations are available:

```
ABS, NEG, ADD, SUB, MUL, DIV, MOD, MULDIV, GETAT,
GETATFIXED, INTERPOLGROUP, SEARCHDISTRIB, GETHIGHPART
```

The "Shift", "Operand1", "Operand2", "Operand3" and "Result" fields are also combo boxes. The entries for all five of these fields are the same, and list all permissible types. These are:

```
ALL, UINT8, UINT16, UINT32, SINT8, SINT16, SINT32
```

The `ALL` type does not stand for any particular type, but is provided as a wildcard. You can also choose not to select a type. This option, however, is only allowed for operations that allow the use of optional parameters.

The **Limitation** and **Rounded** options can be enabled for some operations.

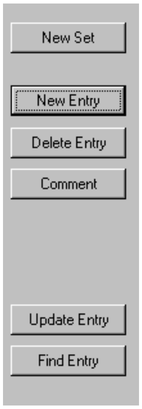
The "Generated Key" field displays the key for the function, which is needed by ASCET (see the section entitled, "Function Key"). This field cannot be edited because these keys can be generated only by the program.

The "Code for representation" field shows the function call used for generating code in ASCET (see the section entitled, "Function Declaration") . In principle, any entry is allowed here because this does not have to follow any particular syntax and cannot be verified for correct syntax by the program.

The "Parameters" field is a combo box where you can select a parameter (such as `%i1%`) from a list. The parameter you select is added to the "Code for representation" automatically.

The Elements of the Button Area:

This area provides the command buttons for the most frequently called-for functions.



- **New Set** – Creates a new (empty) set.
- **New Entry** – Creates a new (empty) entry.
- **Delete Entry** - Deletes the current entry.
- **Comment** – Converts the current entry to a comment.
- **Uncomment** (replaces the **Comment** button if a comment is selected) – Converts the current comment to an entry.
- **Update Entry** - Applies the changes for the current entry.
- **Find Entry** - Searches for an entry.

Note

Buttons can be enabled or disabled based on the current status of the program.

Using the AS Editor

Loading a file:

- Click the **Open** button

or

- select the **File** → **Open** menu command.

  This opens the file selection dialog.
- Select the file you want to open.
- Click the **Open** button to load the file

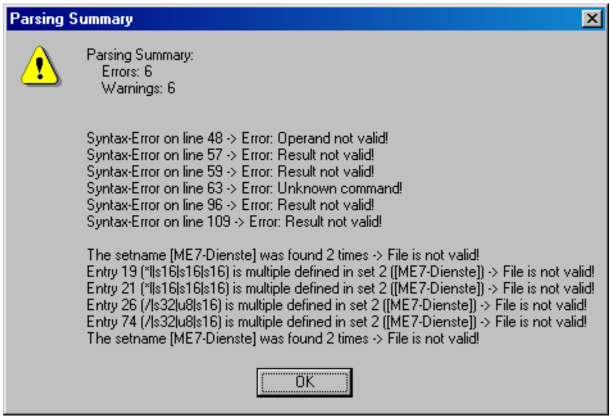
or

- click **Cancel** to cancel the procedure.

Note

You can open one of the four last opened files directly from the **File** *menu.*

When you open a file, the program loads the contents of the file. If, during this process, the program detects errors in entries, it generates messages accordingly and lists the results in the "Parsing Summary" window after it has finished loading the file.



```
Parsing Summary:
   Errors: 6
   Warnings: 6

Syntax-Error on line 48 -> Error: Operand not valid!
Syntax-Error on line 57 -> Error: Result not valid!
Syntax-Error on line 59 -> Error: Result not valid!
Syntax-Error on line 63 -> Error: Unknown command!
Syntax-Error on line 96 -> Error: Result not valid!
Syntax-Error on line 109 -> Error: Result not valid!

The setname [ME7-Dienste] was found 2 times -> File is not valid!
Entry 19 (*l|s16|s16|s16) is multiple defined in set 2 ([ME7-Dienste]) -> File is not valid!
Entry 21 (*l|s16|s16|s16) is multiple defined in set 2 ([ME7-Dienste]) -> File is not valid!
Entry 26 (/|s32|u8|s16) is multiple defined in set 2 ([ME7-Dienste]) -> File is not valid!
Entry 74 (/|s32|u8|s16) is multiple defined in set 2 ([ME7-Dienste]) -> File is not valid!
The setname [ME7-Dienste] was found 2 times -> File is not valid!
```

Entries with errors are converted automatically to comment lines and provided with a description of the error. Such entries as easy to find in the "Entries" field, as they are indicated with a red or yellow LED symbol.

Note

When loading a file, the program ignores all entries that are located above the first set definition (i.e. above the first left square bracket "["). Likewise, all empty lines are ignored and are not loaded.

After all entries have been loaded, those entries belonging to the first set found in the file are listed in the "Entries" field.

Saving a file:

- Click the **Save** button in the toolbar

or

- select **File** → **Save** from the menu to save the current file.

  If you have just created the current file, the standard dialog for saving a file opens.

- If this is the case, select a file name and a directory for the new file.

- Confirm your input by clicking **Save**

or

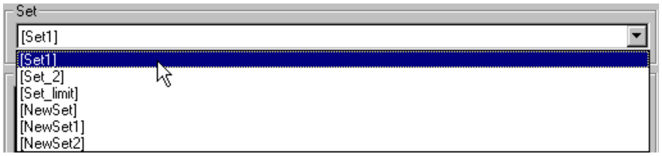
- click **Cancel** to cancel the procedure.

Managing Sets:

After a file has been loaded, all of the sets it contains are listed for selection in the "Sets" combo box.

Selecting a set:

- Click in the "Set" combo box directly, or on the arrow button to its right.
- Select an entry from the drop-down list that appears.



Set

[Set1]

[Set1]
[Set_2]
[Set_limit]
[NewSet]
[NewSet1]
[NewSet2]

If the number of sets it too large to fit within the drop-down list box, a scrollbar appears to the right of the list.

The set you select is activated and its entries are listed in the "Entries" field.

Creating a set:

- Select the **Sets** → **New** menu command

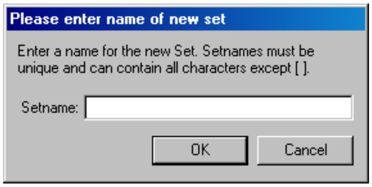
or

- click the **New Set** button in the toolbar

or

- click the **New Set** button in the button area.

  The dialog window entitled, "Please enter name of new set," opens.



Please enter name of new set

Enter a name for the new Set. Setnames must be unique and can contain all characters except [ ].

Setname:

- Enter a name for the new set.

Note

Because the sets are distinguished from each other by their names, each set must be given a unique name.A name must contain at least 1 and no more than 255 characters, and may not contain the [*or*] *characters.*

- Confirm your input by clicking **OK**.

  The new set is appended to the end of the list of existing sets and then automatically selected as the currently active set. The "Entries" field is now empty, as the new set does not yet contain any entries.

Or

- Click **Cancel** to cancel the procedure.

Deleting a set:

- Select the **Sets** → **Delete** menu command

or

- click the **Delete Set** button in the toolbar.

  You are asked to confirm that you want to delete the set and all of the entries that belong to it.

- Confirm to delete the set by clicking **OK**.

Or

- Click **Cancel** to cancel the procedure.

Note

This permanently deletes the set and all of its entries.

Duplicating a set:

- Select **Sets** → **Copy**

or

- click the **Copy Set** button.

  The dialog window entitled, "Please enter name of new set," opens.

- Enter a name for the new set.
- Confirm the procedure by clicking **OK**.

  The duplicate copy of the current set is assigned the new name, appended to the end of the list of existing sets, and then automatically selected as the currently active set.

Or

- Click **Cancel** to cancel the procedure.

Renaming a set:

- Select **Sets** → **Rename**.

Or

- Click the **Rename** button.
- Enter the new name for the set in the "Please enter name of new set" dialog.
- Confirm the procedure by clicking **OK**.

Or

- Click **Cancel** to cancel the procedure.

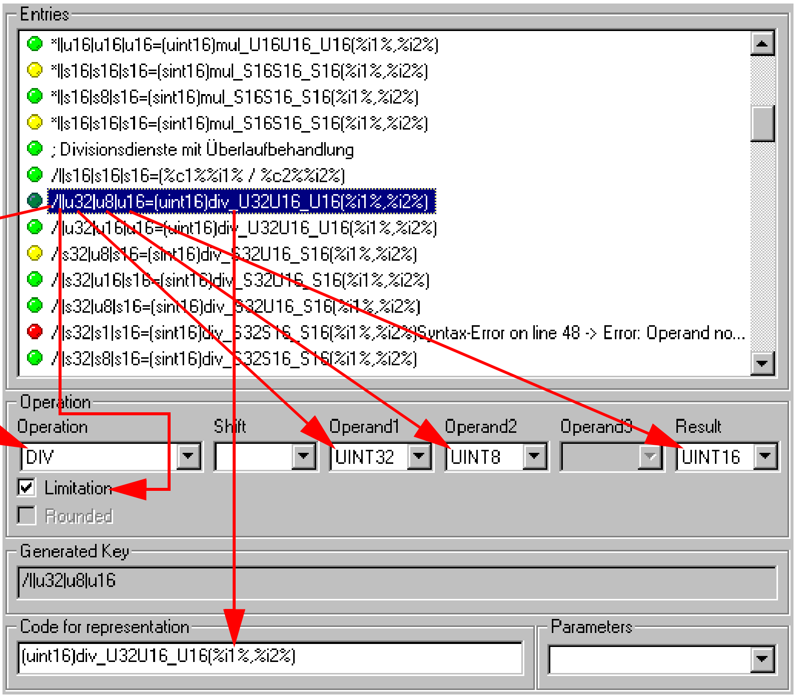
Managing Entries:

If a file has been loaded and a set selected, all of the entries belonging to the current set are displayed in the "Entries" field. If there too many entries to be displayed in the list box, a scrollbar is provided to the right of the field. To view in the *edit area* all of the information pertaining to an entry, simply select one of the listed entries by clicking it with the mouse.

Selecting an entry:

- Click an entry in the "Entries" field.

  The values that belong to the entry are displayed in the edit area.The arrow in the illustration above indicate which component of the entry is represented by which field in the edit area.



Entries

```
● *||u16|u16|u16=(uint16)mul_U16U16_U16(%i1%,%i2%)
● *||s16|s16|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● *||s16|s8|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● *||s16|s16|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● ; Divisionsdienste mit Überlaufbehandlung
● /||s16|s16|s16=(%c1%%i1% / %c2%%i2%)
● /||u32|u8|u16=(uint16)div_U32U16_U16(%i1%,%i2%)
● /||u32|u16|u16=(uint16)div_U32U16_U16(%i1%,%i2%)
● /||s32|u8|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|u16|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|u8|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|s1|s16=(sint16)div_S32S16_S16(%i1%,%i2%)Syntax-Error on line 48 -> Error: Operand no...
● /||s32|s8|s16=(sint16)div_S32S16_S16(%i1%,%i2%)
```

Operation

Operation	Shift	Operand1	Operand2	Operand3	Result
DIV		UINT32	UINT8		UINT16

☑ Limitation
☐ Rounded

Generated Key: /||u32|u8|u16

Code for representation: (uint16)div_U32U16_U16(%i1%,%i2%)

Parameters:

If the entry contains errors (indicated by a yellow or red LED symbol), it is handles automatically like a comment.

Basically, there are two types of entries: functions and comments. Comments always begin with a ";". If the entry you have selected is a comment, all of the fields in the edit area are disabled except the "Code for representation" and "Parameters" fields. If the entry you have selected is a function, fields are enabled and disabled based on the operation and the relevant values are displayed in these fields.

Creating an entry:

- Select **Entry** → **New**.

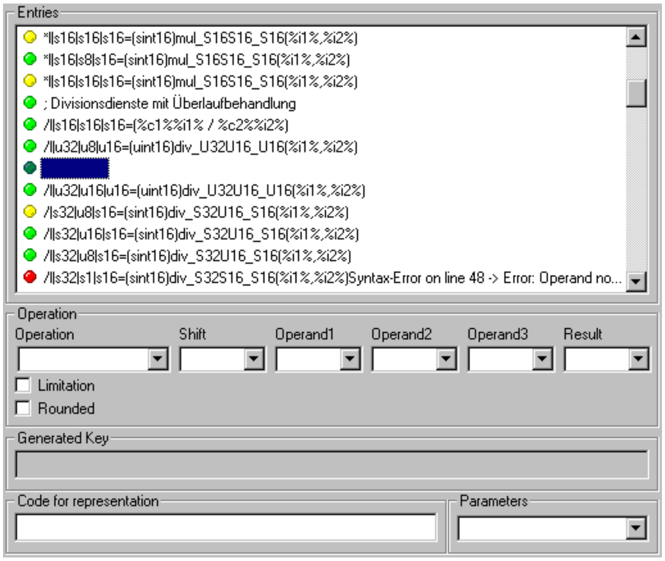
Or

- Click the **New Entry** button in the toolbar.

Or

- Click the **New Entry** button in the button area.

  This creates a new, empty entry in the current set below the entry previously selected, and causes all of the elements of the edit area to become enabled.



Entries

```
● *||s16|s16|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● *||s16|s8|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● *||s16|s16|s16=(sint16)mul_S16S16_S16(%i1%,%i2%)
● ; Divisionsdienste mit Überlaufbehandlung
● /||s16|s16|s16=(%c1%%i1% / %c2%%i2%)
● /||u32|u8|u16=(uint16)div_U32U16_U16(%i1%,%i2%)
●
● /||u32|u16|u16=(uint16)div_U32U16_U16(%i1%,%i2%)
● /|s32|u8|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|u16|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|u8|s16=(sint16)div_S32U16_S16(%i1%,%i2%)
● /||s32|s1|s16=(sint16)div_S32S16_S16(%i1%,%i2%)Syntax-Error on line 48 -> Error: Operand no...
```

Operation

Operation	Shift	Operand1	Operand2	Operand3	Result

☐ Limitation

☐ Rounded

Generated Key

Code for representation

Parameters

The new entry is a function. To create a comment, simply convert the entry you have just created directly into a comment (see "Converting an entry into a comment:" on page 552).

You cannot directly modify entries in the AS editor by editing the character string. Instead, you can influence the individual parameters of the entry by selecting different inputs. To prevent an entry from being modified inadvertently, all changes have to be applied manually by executing an update.

Specifying an entry:

- In the list of entries in the "Entries" field, select the entry you want to change.

  You can edit any of the values in the combo boxes provided in the edit area as needed. To do so, simply select the entries you want from the corresponding combo boxes.
- In the "Operation" combo box, select the desired arithmetic operation.

Note

Depending on the entry you select, various elements in the edit area may be disabled. Those entries which are not allowed for the selected operation are automatically disabled.

- In the "Shift" combo box, select the desired type of shift.

  If you do not want to apply a shift, simply select the blank entry.
- In the "Operand*" " combo box, select the desired types of operators.

  If you select an operator for the access to parameters (`GETAT`, `GETATFIXED`, `INTERPOLGROUP`), the "Operand1" and "Operand2" fields are enabled. If you want to access a *characteristic diagram*, select a value in each of the two fields. If you want to access a *characteristic curve*, leave "Operand2" empty.
- Select the type of result from the "Result" combo box.
- Enable the **Limitation** option if you want to limit the operation.
- Enable the **Rounded** option if you want rounded access to characteristic curves/diagrams.

- Specify a definition of the function or the text for a comment in the "Code for representation" field

or

- delete the present content of the field, if there is any, to create a standard function definition (see page 551).
- When you need further parameters, you can select them one by one from the "Parameters" combo box.

  The selected parameter is inserted at the end of the string in the "Code for representation" field.

After you have completed all necessary changes, you have to update the entry (see "Updating an entry:" on page 550).

Note

The entry in the "Generated Key" field cannot be edited. This value is generated by the program automatically as soon as the changes have been applied.

Updating an entry:

- Select **Entry** → **Update**

or

- click the **Update Entry** button in the toolbar.

or

- click the **Update Entry** button in the button area.

  This applies the values from the edit area and initiates a syntax check. If all of the inputs are correct, the changes are applied to the current entry.

When entries are updated to apply new changes, the program automatically runs a check on the values. This consists of the following criteria:

- complete input,
- correct syntax of the input,
- check for duplicates within the current set.

The changes are only applied if all input is complete and syntactically correct and there is not a duplicate entry for this function key present in the set already. Otherwise, an error message is generated and the entry is left unchanged.

If you leave the "Code for representation" field blank, the program derives a standard function definition from the other input data. This definition should be considered a suggested definition, which can be replaced by another at any time.

Note

If the user manually specifies the input in the "Code for representation" field, the input is not checked. In all cases, the user is solely responsible for ensuring that the functional definition specified is also actually available when code generation is carried out using ASCET!

A list of code that can be generated by the editor is provided in the section entitled, "Allowable Arithmetic Services". The following example is intended to better illustrate the process:

Function key:	**Generates standard function definition:**
`+l|*|*|*`	`addl_%t1%%t2%_%tr%(%i1%, %i2%)`
`+l|*|u8|*`	`addl_%t1%u8_%tr%(%i1%, %i2%)`
`+l|u16|u8|*`	`addl_u16u8_%tr%(%i1%, %i2%)`
`+l|u16|u8|u32`	`addl_u16u8_u32(%i1%, %i2%)`

The user can change how the editor generates function definitions. To do so, you have to edit both the `operations.ini` and `type.ini` files, which are located in the same directory as the editor (by default, `..\ETAS\Ascet5.2\ASEditor`). In these files, a character string can be defined for each operation and each type. These will then be used by the editor in place of the standard strings.

Note

When editing the `operations.ini` *and* `type.ini` *files, be careful not to change the file names, as the AS editor would otherwise not be able to find them and would resort to using the standard settings.*

Example: In the `operations.ini` file, the entry,

```
add = add_
```

is replaced by

```
add = doAddition_
```

Now the code for the key,

```
+|*|*|*
```

will no longer be

```
add_%t1%%t2%_%tr%(%i1%, %i2%)
```

but instead it will be

```
doAddition_%t1%%t2%_%tr%(%i1%, %i2%)
```

Deleting an entry:

Note

*The delete process does **not** provide a prompt for confirmation. An entry that has been deleted cannot be restored. Use this function with caution.*

- Select the **Entries** → **Delete** menu command.

Or

- Click the **Delete Entry** button.

  This deletes the selected entry without any further prompts.

When an entry is deleted, the selection automatically moves down to the next entry in the list. If the deleted entry was the last in the list, the previous entry is selected. This allows you to delete several entries quickly in succession.

The AS editor provides the ability to insert comments in an AS file, for example to provide short descriptions or explanations of the individual entries. This function can also be applied, however, in order to convert existing entries into comments. This simply entails inserting a ";" in front of the actual entry. Both ASCET and the AS editor handle lines preceded with ";" as comments.

Converting an entry into a comment:

- Select an entry in from the list of entries in the "Entries" field.
- Select the **Entries** → **Comment** menu command.

Or

- Click the **Make Comment** button in the toolbar.

Or

- Click the **Comment** button in the button area.

  The selected entry is converted into a comment.

In certain circumstances, comments can be converted to entries. This is possible only if the comment, without its preceding ";", would represent a valid entry. This function can also be used to correct entries that contain errors.

If you choose to convert a comment into an entry, the text contained in the "Code for representation" field is again checked by the parser. If the parser does not detect any errors, the comment is converted into an entry. On the other hand, if the text found in the field does not correspond to a valid entry, the comment is left as a comment (i.e. an entry with an error remains an entry with an error).

Converting a comment into an entry:

- Select a comment in the "Entries" field.

  The name of the **Comment** button in the button area now toggles to **Uncomment**.

- Select the **Entries** → **Comment** menu command.

Or

- Click the **Uncomment** button in the button area.

  The selected comment is converted into an entry, *if it is syntactically correct*. If it is not syntactically correct, no changes are made.

Searching for an entry:

Sometimes you may need to find a particular entry within a set. To do so, it is possible to search for an entry by its unique function key.

- In the edit area, select the values of the function you are looking for in the "Operation", "Operand1", "Operand2", "Operand3", "Shift" and "Result" fields.
- Set the **Limitation** and **Rounded** options to match those of the function you are looking for.
- Select **Entries** → **Find**

or

- click the **Find Entry** button in the button area.

  If the values you have specified match those of an existing entry, the entry is selected automatically and displayed in the "Entries" field.

If no matching function exists, or if the matching function is already selected, a corresponding message is displayed.

5 Signals and Icons

This section describes how to work with signals and icons in the ASCET database. Signals and icons are supplementary data for a control system that can be stored together with the system itself in the ASCET database.

Both signals and icons can be imported into the database or exported to the file system. Signals are used by the system, icons can be assigned to elements and hierarchies. The following sections explain how to work with each type of item.

5.1 The Signal Viewer

With the signal viewer signals can be loaded into ASCET database and viewed. The viewer can read a variety of signal formats and convert signals between formats.

The following measurement data formats are supported by ASCET:

- Tab delimited ASCII. Almost every spreadsheet or database can read and write in this format.
- ETAS Format. This is a highly efficient binary format that is used only internally by ASCET.
- FAMOS Channel Format.
- FAMOS Record Format.
- Matlab Format. This is the MATLAB/Simulink ASCII format.
- MDF Format.

The last four are third-party formats for measurement data. Please refer to the relevant documentation for details. Internally ASCET can process ASCII, ETAS and both types of FAMOS format. MDF and Matlab can be read and written.

To create and open a signal:

- In the Component Manager, select **Insert** → **Signal**.

  A new signal is created.

- Double-click on the signal

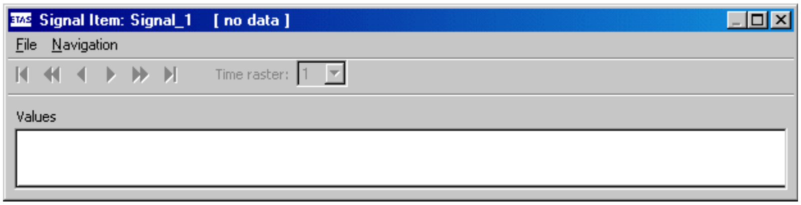
or

- select **Component** → **Edit Item**

or

- press <RETURN>.

  The signal viewer opens.



To import measurement data to a signal:

- Open the signal in the signal viewer.
- In the signal viewer, select **File** → **File In Data** → ***<format>*** to import the signal data from the file system.

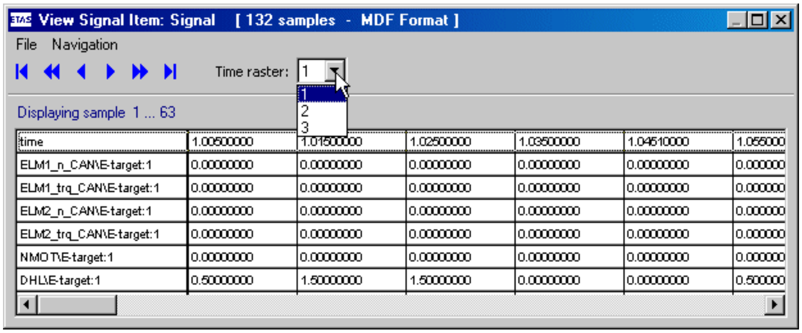
  The Windows file selection dialog box opens.

- Select the path and file name of the data file you want to import.

Note

Here, you can use measurement data you have created and saved during an experiment before.

The data is imported and automatically converted to the specified format. After conversion the data is displayed in the viewer.


View Signal Item: Signal [132 samples - MDF Format]

Displaying sample 1 ... 63

time	1.00500000	1.01500000	1.02500000	1.03500000	1.04510000	1.055000
ELM1_n_CAN\E-target:1	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.000000
ELM1_trq_CAN\E-target:1	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.000000
ELM2_n_CAN\E-target:1	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.000000
ELM2_trq_CAN\E-target:1	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.000000
NMOT\E-target:1	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.000000
DHL\E-target:1	0.50000000	1.50000000	1.50000000	0.00000000	0.00000000	0.500000

When you selected the MDF or FAMOS format for the import, you can select and view different time frames.

To view measurement data:

- Choose **Navigation** → **Show Next** to move to the next data column.

Or

- Click the **Show Next Column** button.
- Choose **Navigation** → **Fast Forward** to move to the next screenful of data.

Or

- Click the **Fast Forward** button.

  If, for instance, the screen shows five data columns, the next five columns are shown.

- Choose **Navigation** → **Show Last** to move to the last data column.

Or

- Click the **Show Last Column** button.

  For each of these three commands there is an equivalent command or button for moving in the opposite direction.

5.2 The Icon Editor

Icons can be assigned to the layout of components. The icon is shown in the graphical block, when the component is included in other components in the block diagram editor. A selection of icons is provided with ASCET, and you can also create your own.

To start the icon editor:

- In the Component Manager, select **Insert** → **Icon** to create a new icon.

Or

- Select an existing icon from the database.
- Double-click on the selected symbol

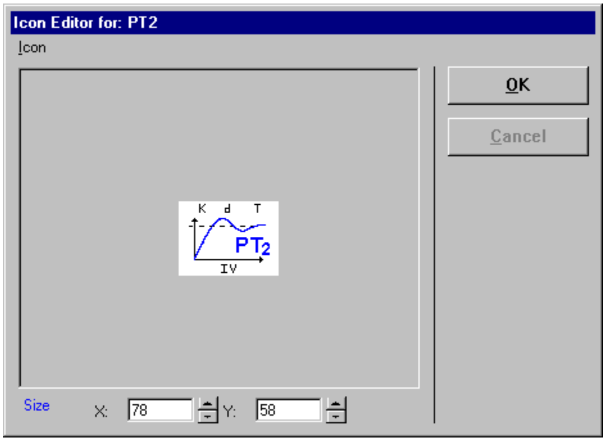
or

- select **Component** → **Edit Item**

or

- press <RETURN>.

  The icon editor opens.



If you have created a new icon, the drawing area is empty. If you have opened an existing icon, it is displayed in the drawing area.

You can load any bitmap file (BMP, PCX and TIFF) and use it as an icon in ASCET. Thus, you can either design your own icons with a drawing program, or use existing ones. You should make sure that the bitmap files you are importing are of an appropriate size, i.e. no larger than the graphical block the icon is to be assigned to. One grid unit in ASCET corresponds to 5 pixels.

To load an icon:

- In the icon editor, select **Icon** → **Load**.

  The "Image File" dialog box opens.

- Select the name of the icon file you want to load.
- Click **OK**.

  The selected icon is displayed in the icon editor.

Icons can be scaled up or down to make them fit the block in which they are to be used. Scaling will, however, result in a degradation of image quality. Scaling by large factors should therefore be avoided.

To scale an icon:

- Open the icon editor for the icon you want to scale.
- Adjust the vertical and horizontal size of the icon with the two "Size" boxes underneath the icon pane.
- The icon size changes immediately, each time you adjust a value in the "Size" boxes.
- Click **OK**.

It is possible to save ASCET icons as bitmap files. This is useful, for instance, if you want to illustrate the documentation of your components.

To save an icon to a file:

- Open the icon editor for the icon you want to store to a file.
- Select **Icon** → **File Out** → ***<file format>***.

  The "Image File" dialog window opens.
- Select the filename and the path where you want to save the icon.
- Click **OK**.

  The icon is stored in the appropriate format (BMP, PCX or TIFF).

6 Experimentation

6.1 The Experimentation Environment

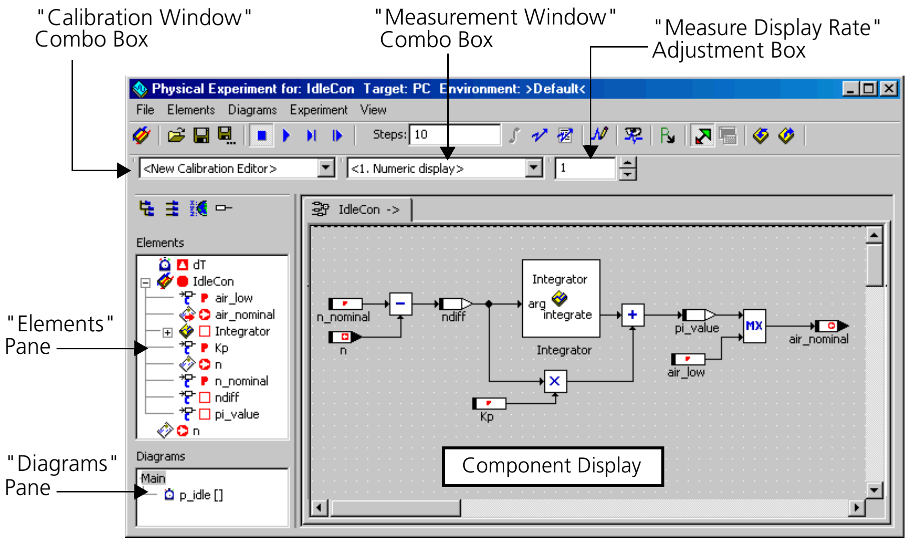
ASCET supports a highly modular approach to software development. The individual components of a project can be developed independently of each other, and combined after they have been tested thoroughly. ASCET provides the experimentation environment for testing components and projects.

Projects can be tested in offline and online experiment. If the project is connected to any external hardware, that connection can only be tested in an online experiment.

Note

Online experiments are only possible when ASCET-RP is installed. Therefore, online experimentation is described in detail in the ASCET-RP user's guide.

With the experimentation environment it is possible to stimulate (in an offline experiment) the processes and methods, as well as the elements of a component or project. The data values of the elements can be measured in various visualizations, written to a file, and calibrated interactively. The main elements of the experiment window are shown in the following illustration (showing an offline experiment):



The elements of the component are listed in the "Elements" pane to the left of the component display. The component display shows the block diagram, if the component was specified as a block diagram. For components specified in C code or ESDL, the code is shown. For projects, the project editor tabs are shown.

Beneath the "Elements" pane is the "Diagrams" pane, which lists all diagrams of the current component. The "Measurement Window" combo box is underneath the button bar. It lists all measurement windows available. To the left of the "Measurement Window" combo box is the "Calibration Window" combo box, which lists all available calibration windows. The title bar shows the name of the current component or project and the current target.

6.1.1 Description of the Menu Options

- **File:**
  - *Load Environment*

    Loads an environment (configuration) for the current experiment.
  - *Save Environment*

    Saves the current experiment configuration.
  - *Save Environment As*

    Saves the current experiment configuration with a different name.
  - *Export Environment*

    Exports the experiment configurations of the current component.
  - *Exit*

    Leaves the Offline experimentation environment.
- **Elements:**
  - *Show Component*

    Displays the selected referenced component.
  - *Calibrate (<*CTRL*> + <*C*>)*

    Opens a calibration window for the selected elements.
  - *Stimulate (<*CTRL*> + <*S*>)*

    Note

    This menu option is only available in offline experiments.

    Opens the "Stimulus" dialog for the selected element.

- *Measure (<CTRL> + <M>)*

  Opens a measurement window for the selected elements.
- *Log (<CTRL> + <L>)*

  Adds selected elements to the Data Logger.
- *Log All (<CTRL> + <A>)*

  Adds all elements to the Data Logger.
- *Reinitialize*

  Reads the values of certain elements from the current data set of the component;
  → *Variable*s – of variables,
  → *Parameters* – of parameters,
  → *Both* – of variables and parameters.
- *Write Back Data*

  Writes the values of the selected elements to the current data set of the component.;
  → *Selected Element*s – selected elements,
  → *Calibrated Elements* – elements calibrated during the experiment.
- *Show Element Implementation (<CTRL> + <I>)*

  Shows the implementation of the selected element (read only).
- *Update Dependent Parameters (<CTRL> + <U>)*

  Updates the values of dependent parameters.
- *Load Data*

  Loads data from a file. The options for this procedure are set in a special window.
- *Save Data*

  Writes the data of selected parameters or variables to a file. The options for this procedure are set in a special window.

• **Diagrams:**

- *Show Diagram*

  Displays the selected diagram.

– *Stimulate*

> **Note**
>
> *This menu option is only available in offline experiments.*

Opens the "Event" window for the selected process or method. The event can be set up.

- **Experiment:**

> **Note**
>
> *These menu options are only available in **offline** experiments.*

– *Event Generator → Open*

Opens the event generator.

– *Event Generator → Step Mode*

Sets the step mode type:
→ *Steps* – single step mode (default),
→ *Timed [s]* – timed step mode,
→ *Break At Condition* – breakpoint step mode.

– *Event Generator → Edit Break Condition*

Opens the breakpoint condition editor.

– *Event Tracer*

Opens the "Event Tracer" window for data tracing.

– *Data Generator*

Opens the data generator.

– *Data Logger*

Opens the data logger.

– *Automatic Stop*

Stops the experiment automatically when the end of the stimulating signal is reached (cf. page 581).

– *Stop Experiment*

Stops the experiment.

– *Start Experiment*

Starts the experiment.

– *Pause Experiment*

The experiment is paused.

- *Step Experiment*

  The experiment is performed in step mode.
- *Open Target Debugger*

  Opens the debug window for C code components.
- *Update Calibration Windows*

  Updates the contents of the calibration windows.
- *Close Calibration Windows*

  Closes all open calibration windows.
- *Close Measure Windows*

  Closes all open measurement windows.

- **View:**
  - *Redraw*

    Redraws the block diagram.
  - *Hide Seq. Calls*

    Hides the sequence calls of the displayed block diagram.
  - *Show Seq. Calls*

    Shows the sequence calls of the displayed block diagram.
  - *Monitor All*

    Assigns monitors to all elements.
  - *Delete Monitors*

    Deletes all monitors in the diagram.
  - *Show Parent Component*

    Returns to the previous component.

6.1.2 Opening and Setting Up the Experimentation Environment

You can start the experimentation environment for a component or project either from within the respective component editor or the project editor.

To open the experimentation environment for an offline experiment:

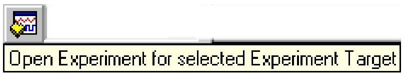
- Open the appropriate component or project.

- If you want to experiment offline with a project, select the target **PC** from the build options.

  The "Experiment Target" combo box now contains the entry `Offline (PC)`.
  For components, the combo box is disabled.

- Select **Component** → **Open Experiment**

or



- click on the **Open Experiment for selected Experiment target** button.

  The default experimentation environment opens for the component or project. If more than one environment has been stored, you can choose which one to open. For details see section "Loading and Saving Environments" on page 594.

When experimenting with components, the global elements (see "Defining Global Communication" on page 388) are under some circumstances not updated properly in the default project. The following error message is displayed in the ASCET monitor window:

```
Error: need export for import element <name> with type
<type>
```

To correct this error, proceed as follows:

Define global elements in the default project:

- In the component editor, select **Component** → **Default Project** → **Resolve Globals** to resolve the global elements.

  You can now restart the experiment.

Now you can set up the experiment. Setting up an offline experiment consists of four steps:

- Starting the experimentation environment from the Component Manager or the respective component editor. The experimentation environment works the same for all types of component.
- Setting up the event generator. The event generator determines which methods or processes are triggered in which mode. When a project is experimented with offline, the tasks are stimulated, rather than the methods or processes.

- Setting up the data generator. Here some elements of the component are stimulated with a configurable flow of data.
- Setting up the measurement and calibration windows. The values of all elements can be displayed in a variety of forms, e.g. numerically or in an oscilloscope.

6.1.3 The Event Generator

In an online experiment the various tasks and processes of a project are scheduled by the real-time operating system. During offline experiments the scheduling is simulated by the event generator. The event generator determines which methods or processes of the component under experimentation are activated in which order and in which mode. An event has to be defined for each method or process that is to be activated.

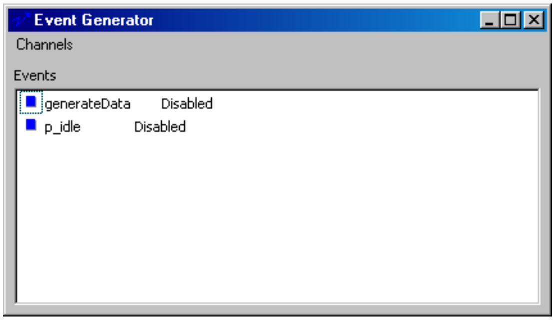
To set up the event generator:

- Select **Experiment** → **Event Generator** → **Open**

or

- click on the **Open Event Generator** button.

  The "Event Generator" window opens. It contains an event for every method (for classes) or process (for modules) defined in the component. By default, all events are disabled, so you have to enable the ones you want to use in the experiment.



generateData Disabled
p_idle Disabled

- Select an event.
- Select **Channels** → **Enable**

or

- right-click on the event and choose **Enable** from the context menu.
- Repeat these steps for each event you want to enable.
- Select **Channels** → **Enable** again to disable an event again.

Methods or processes for which no event has been enabled are not activated and therefore will have no influence on the experiment.

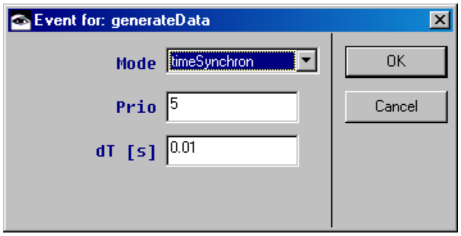
In addition to the events for the methods and processes, a `generateData` event is always created by default. This event triggers the generation of data that have been defined in the data generator. If your experiment does not require any data to be generated, you can leave this event disabled, otherwise it must always be enabled.

To set up an event:

Once you have created an event, it is assigned default values for all the event options. It may not always be necessary to edit these options.

- Select the event you want to set up.
- Select **Channels** → **Edit**.

  The "Edit Event" dialog box appears.


Event for: generateData

Mode: timeSynchron

Prio: 5

dT [s]: 0.01

- Adjust the event options.

  The meaning of the various options is explained on page 569.
- Click **OK**.
- Repeat for each event you want to set up.

To set up an event directly:

- In the "Diagrams" pane, select the process or method for which you want to generate an event.

- Right-click on the process or method and choose **Stimulate** from the context menu.

Or

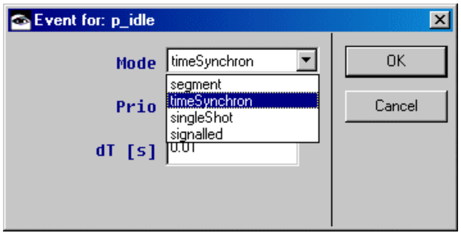
- Select **Diagrams** → **Stimulate**.

  The event is enabled and the "Event" dialog box is opened for the event. Using this command is equivalent to first enabling the event in the "Event Generator" window and then editing it.

Event Options

The "Event" dialog box offers various options for every event.

- Every event has a *mode*. There are four modes:



  - A *time synchronous event* (`timeSynchronous`) is triggered once at the beginning of every interval. The length of an interval is determined by the value set in the "dT [s]" field.
  - A *segment event* (`segment)` is used in automotive application where the triggering of the event depends on the rotational speed of the engine, see "To set up a segment event:" for details.
  - A *single event* (`singleShot)` is triggered only once, when the simulation is started. This is useful e.g. for initialization methods. You can re-trigger the *singleShot* event, by choosing **Channels → Reactivate Event** in the "Event Generator" window.
  - An *asynchronous event* (`signalled`) is stimulated with the data of a real measurement. Thus, real-world data can be used as trigger even during offline experimentation. Setting up an asynchronous event is described on page 571.

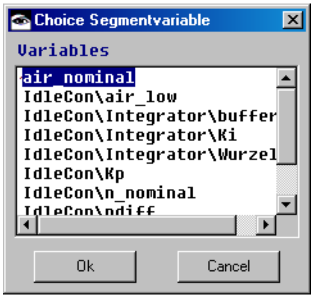
- The *priority* of an event determines the order in which events are calculated. Often several events are assigned to the same time frame, e.g. 10 milliseconds. In that case the event with the highest priority (i.e. the highest number) is triggered first, the other ones are triggered in turn.

- The *dT* value of an event determines the interval in which it is triggered. If the `dT` value is 0.01 seconds, the event is triggered every 10 milliseconds. The smallest possible `dT` value is one microsecond (10^{-6} seconds).

To set up a segment event:

- Open the "Event" dialog box for the event.
- Select `segment` from the "Mode" combo box.

  A dialog box opens that lists all variables of your component.

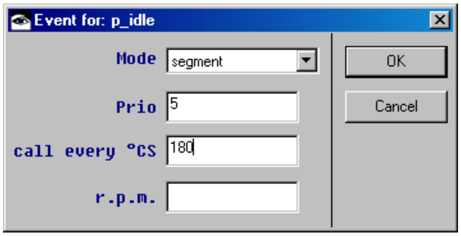

Choice Segmentvariable

Variables

```
air_nominal
IdleCon\air_low
IdleCon\Integrator\buffer
IdleCon\Integrator\Ki
IdleCon\Integrator\Wurzel
IdleCon\Kp
IdleCon\n_nominal
IdleCon\ndiff
```

- Select the variable that is to serve as the segment variable and click **OK**.

  The segment variable should represent the rotational speed in revolutions per minute.
- Adjust the crankshaft angle in the "call every °CS" field of the "Event" dialog box.
- Click **OK**.


Event for: p_idle

Mode: segment

Prio: 5

call every °CS: 180

r.p.m.:

The segment event is now triggered at the beginning of every segment interval `tseg` (in degrees), which is calculated according to the following formula:

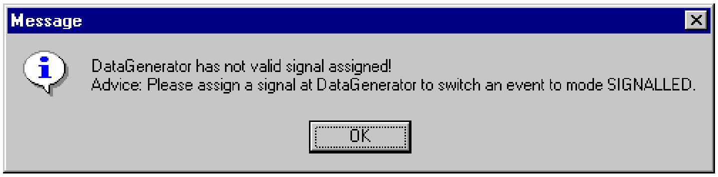
$$t_{seg}[s] = \frac{CS[deg]}{n\left[\frac{1}{min}\right] \cdot 6}$$

n is the rotational speed in revolutions per minute, and `CS` is the crankshaft angle that specifies the revolution.

To set up an asynchronous event:

- Open the "Event" dialog box for the event.
- Select `signalled` from the "Mode" combo box.

  If no signal has been selected in the data generator (see "The Data Generator" on page 573), the following error message is displayed:

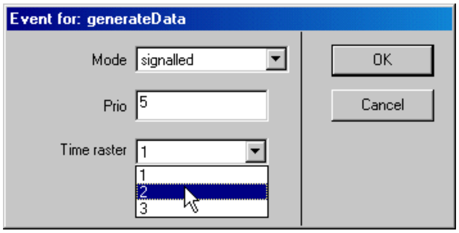


Message

DataGenerator has not valid signal assigned!
Advice: Please assign a signal at DataGenerator to switch an event to mode SIGNALLED.

- To remove the error, proceed as follows:
  - Confirm the error message with **OK**.
  - If you set up the event directly (cf. page 568), close the "Event" window.
  - In the "Physical Experiment" window, click on the **Open Data Generator** button to open the data generator.
  - Define a signal as described in "To define a signal:" on page 578.

– Repeat the first two steps of the instruction.


Event for: generateData

Mode: signalled

Prio: 5

Time raster: 1

– Choose the desired time raster, if the signal contains multiple rasters.

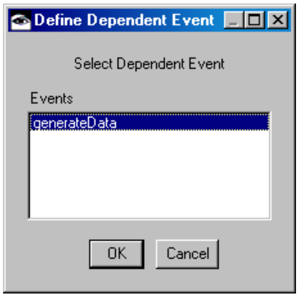
- Click **OK** to accept the settings.

A dependent event is activated whenever the event it depends on is activated. This does not affect the other settings of the event. It is still activated normally in addition to being activated as a dependent event, but the activation is synchronized with the event it depends on.

To set up a dependent event:

- Select an event in the event generator window.
- Choose **Channels** → **Dependent Event**.

  The "Define Dependent Event" dialog box opens.



- Select the event the first event is to depend on.
- Click **OK**.

6.1.4 The Data Generator

In an offline experiment, the data generator provides the stimuli for the elements of the component being experimented with. Usually the elements that are stimulated are the interface elements of the component, but any element can be stimulated with the data generator. The data generator provides a number of different stimulus modes, such as sine waves, pulses, etc.

Note

In offline experiments, you often stimulate variables and parameters that use actual model values when the experiment runs in online mode. You may need to adjust your experiment settings accordingly.

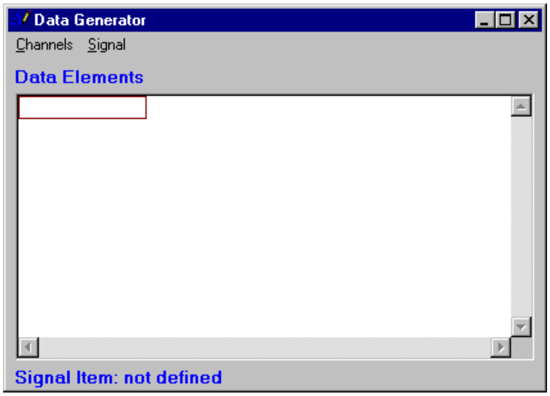
To set up the data generator:

- In the experimentation environment, click on the **Open Data Generator** button

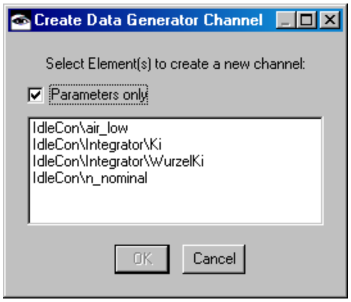
or

- choose **Experiment** → **Data Generator**.

  The "Data Generator" window is opened.



- Select **Channels** → **Create** to open the corresponding dialog box.


Create Data Generator Channel
Select Element(s) to create a new channel:
☑ Parameters only
OK
Cancel

IdleCon\air_low
IdleCon\Integrator\Ki
IdleCon\Integrator\WurzelKi
IdleCon\n_nominal

By default, the dialog lists only the parameters for the current component. You can stimulate all basic elements for a component.

- Deactivate the **Parameters only** option to view all basic elements for the component.
- Select the element or elements for which you want to create a channel.

  You can select more than one element by clicking on them while holding down the <CTRL> key.
- Click **OK**.
- Repeat for the other elements for which you want to create data channels.

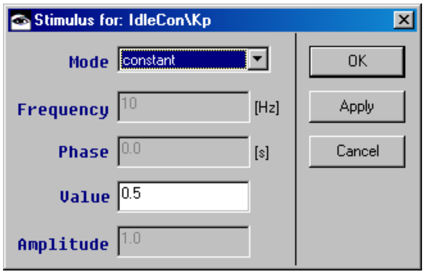
To set up a channel in the data generator:

- Select a channel in the "Data Generator" window.
- Select **Channels** → **Edit**

or

- double-click on the selected channel.

  The "Stimulus" dialog box opens. It shows the name of the channel being set up in its title bar.


Stimulus for: IdleCon\Kp

Mode: constant

Frequency: 10 [Hz]

Phase: 0.0 [s]

Value: 0.5

Amplitude: 1.0

- Alternatively, select an element from the "Elements" list in the experimentation environment.
- Select **Stimulate** from the context menu

or

- select **Elements** → **Stimulate**.

  This adds the element to the "Data Generator" window and opens the "Stimulus" dialog box.
- Select a mode for the channel. The mode determines what kind of curve is generated. Possible modes are `constant`, `sinus`, `ramp`, `pulse`, `step`, `table`, `matrix`, `random` and `gaussian`.

To set up a stimulation mode:

- Perform the necessary steps for each channel in the data generator.

Note

You can modify the settings for a signal channel while your experiment is running. To test different settings, simply press **Apply** *to change the signal without closing the dialog.*

1. `constant` mode

   - In the "Stimulus" dialog, insert the value of the constant in the "Value" field.

     All other setup fields are deactivated.
   - Click **OK** to assign the value and close the window.
   - Click **Apply** to assign the value without closing the window.
   - Click **Cancel** to discard the setting and close the window.

2. cyclic modes (`sinus`, `ramp`, `pulse` und `step`) and random generator (`random`; equal distribution)

   - Set the frequency for the wave.
   - Set the phase for the wave.

     The phase determines the offset from the time axis. If, e.g. the phase of a sine-signal is 0.5 seconds, the signal will be shifted to the right by half a second.

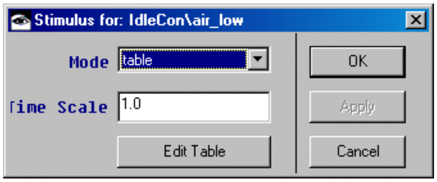
   Note

   The "Frequency" and "Phase" fields are not available for `random` *mode.*

   - Set the offset of the y-axis (`Offset`).
   - Adjust the amplitude.
   - Click **OK** or **Apply**.

3. `table` mode

   The `table` mode uses a table as stimulus.


Stimulus for: IdleCon\air_low

Mode: table

Time Scale: 1.0

- Klick on the **Edit Table** button to edit the table used as stimulus.

  The table editor for one-dimensional tables opens.

- In the table editor, enter the desired values directly

or

- select **Edit** → **File In Data** to load an existing table.

  Details about working with the table editor are given in sections "The Editor for Combined Types (Table Editor)" on page 460 and "Working with Calibration Windows" on page 620.

- In the "Time Scale" field, enter a factor for the time scale.

  The X axis values of the table are multiplied with this factor, and the results are used as time steps (in seconds) for stimulation.

- Click **OK** to accept the settings.

Note

The table is processed only once. If you want to use it a second time to stimulate a channel, you have to restart the experimnent.

4. `matrix` mode

   The setup of the `matrix` mode is described in the two sections on page 578.

5. `gaussian` mode

   This mode is only available for numerical variables. It stimulates the variable using a random number generator with Gaussian distribution.

   - In the "Mean" field, enter the mean value of the Gaussian distribution.
   - In the "Variance" field, enter the variance of the Gaussian distribution.
   - Click **OK** or **Apply**.

To delete channels:

- Choose **Channels** → **Delete** to delete the selected channel from the data generator.
- Choose **Channels** → **Delete All** to delete all channels from the data generator.

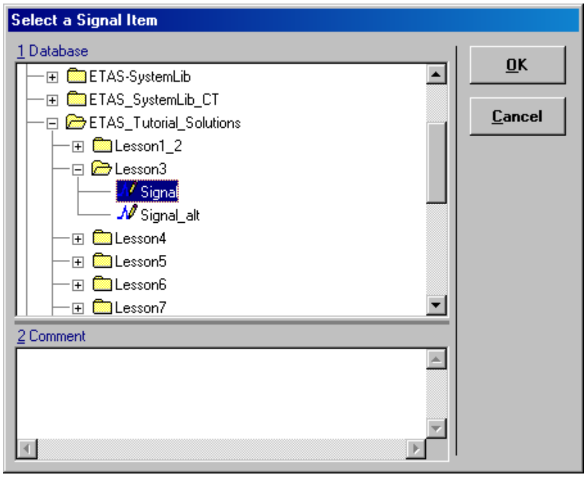
It is possible to use the data of actual measurements as a stimulus in the data generator. That way, components can be tested with real-world data even during offline experimentation. Such data is stored in signal items in the database and can be read in from a variety of formats. Importing signals is described in chapter "Signals and Icons" on page 555. Individual channels from a signal can be assigned as channels in the data generator.

To define a signal:

You can only define a signal when the experiment is stopped. Proceed as follows:

- In the data generator, select **Signal** → **Define Signal** .

  The "Select a Signal Item" dialog box opens.



- Select a signal from the "1 Database" pane.
- Click **OK**.

  The signal is now defined. You can have only one signal item per experiment.

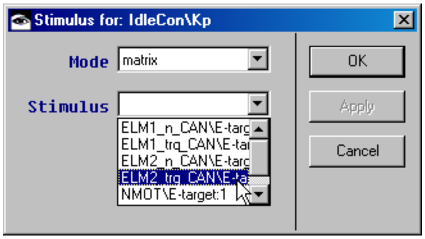
To define data generator channels as signal channels:

- In the data generator window, select a channel.
- Choose **Channels** → **Edit**

or

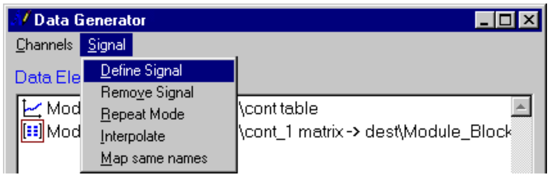
- double-click on the channel.
- Choose `matrix` from the "Mode" combo box in the "Stimulus" dialog box.

  The "Stimulus" combo box appears underneath the "Mode" combo box. It lists all channels contained in the signal defined earlier.



- Select the signal channel you want to assign to the data generator channel.
- Click **OK**.

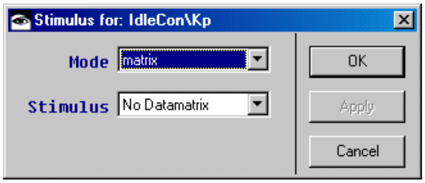
When you have defined a signal (cf. page 578), new menu items appear in the **Signals** menu in the data generator. However, these are only available when the experiment is stopped.



To remove a signal:

- In the data generator, select **Signal** → **Remove Signal**.

  The defined signal is removed. A channel stimulated with the signal keeps the `matrix` mode, but the entry in the "Stimulus" combo box is reset.



When the signal channels and the data generator channels are named equally, you do not have to select each channel separately in the "Stimulus" combo box (cf. page 579). You can assign channels with identical names automatically.

To assign channels with identical names:

- In the "Stimulus" dialog, select the `matrix` mode for each desired channel.
- In the data generator, select **Signal** → **Map same names**.

  Signal channels are assigned to data generator channels with identical names.

Each time data is generated in the experiment (cf. page 568), the signal is evaluated. If the actual time stamp falls between two signal points, the lower signal value is assigned to the channel by default. As an alternative, you can select linear interpolation.

To interpolate the signal:

- In the data generator, select **Signal** → **Interpolate**.
- Start the experiment.

  The assigned value is linearly interpolated from the signal points enclosing the actual time stamp.

If nothing else is specified, the experiment continues after the end of the signal is reached. The stimulated channel retains the last signal value. You can select a repeat mode for the signal or an automatic stop of the experiment.

Note

The automatic stop overrules the repeat mode; when both are selected, the experiment stops at the end of the signal..

To select signal repetition:

- In the data generator, select **Signal** → **Repeat Mode**.
- Restart the experiment.

  When the end of the signal is reached, stimulation begins anew.

To set up the automatic stop:

This command is available only when at least one data generator channel is stimulated with a signal.

- In the "Physical Experiment" window, select **Experiment** → **Automatic Stop**.

  The menu function is marked. When you start the experiment now, it will automatically stop when the end of the signal is reached.
- Select **Experiment** → **Automatic Stop** once more to deactivate the automatic stop.

The setting of **Experiment** → **Automatic Stop** is kept when the experiment is ended, and the experiment environment is closed. The next time you experiment with this component, the old setting is assumed.

6.1.5 The Measurement System

The measurement system offers a variety of ways of displaying the values of elements, such as an oscilloscope, bar displays and numerical displays. Each measurement window has to be set up with the values it is to show. The displays offer a variety of customizing options which can be adjusted during setup of the experiment, but also while it is running. All measurement windows are opened by assigning elements to them from the experimentation environment. The measurement windows are described in chapter "Measurement Windows" on page 648.

To assign an element to a new measurement window:

- Select the type of measurement window that you want to assign the element to from the "Measurement Window" combo box.

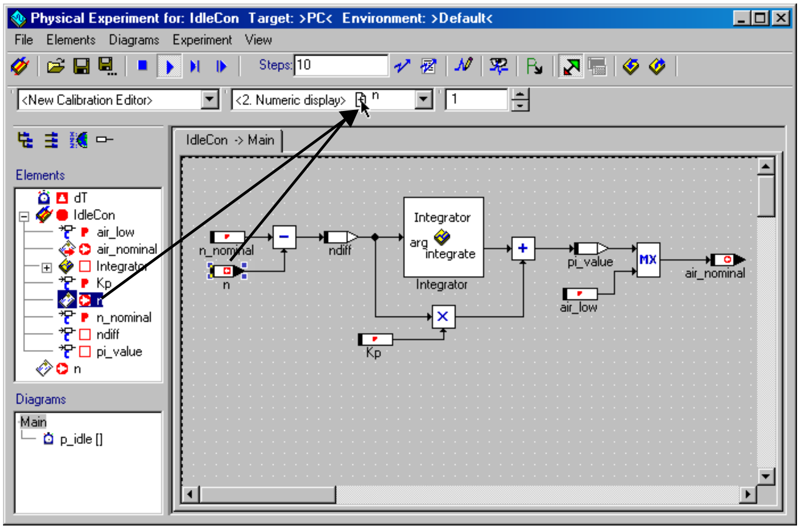
  The available types are listed in brackets, e.g. `<1. Oscilloscope>`.
- Select the element that you want to assign to a measurement window in the "Elements" pane.
- Select **Elements** → **Measure**.

Or

- Drag the element from the "Elements" pane to the "Measurement Window" combo box.

Or

- Drag an occurrence of the element from the block diagram display to the "Measurement Window" combo box.



A new measurement window is opened and the selected element is added to the measurement window. You can select more than one element by clicking on them while holding down the <CTRL> key, in that case all selected elements are assigned to the window.

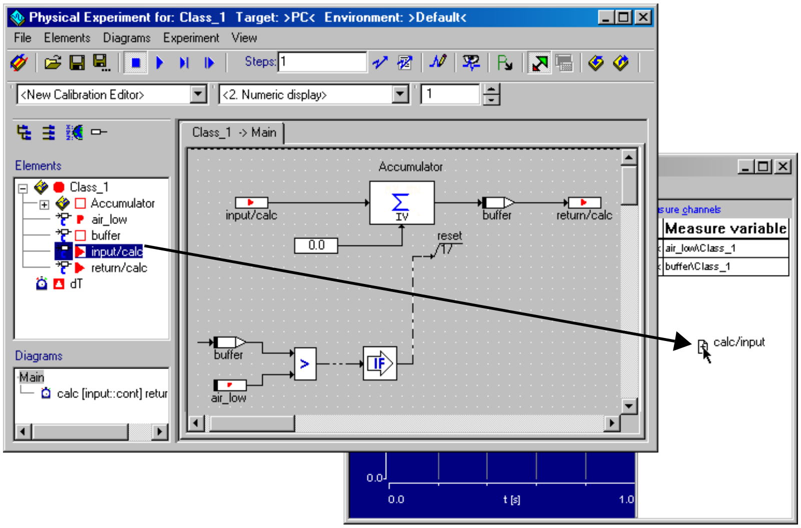
The "Measurement Window" combo box also shows the titles of all measurement windows that are already open. The entries for open windows are displayed without brackets, e.g. `Oscilloscope; 1`. If you have changed the title of a measurement window, that title is listed in the combo box.

To assign an element to an existing measurement window:

- Select the entry for the existing measurement window from the "Measurement Window" combo box.
- Drag one or more elements to the "Measurement Window" combo box.

Or

- Drag the elements to the measurement window.



The monitor displays the current value of the element above the selected occurrence in the drawing area. For details see section "Monitor" on page 677.

To assign a monitor to an element:

- Right-click an occurrence of an element in the drawing area.
- Select **Monitor** from the context menu.

  The current value of the element is shown in the component display.

To close all measurement windows:

- Select **Experiment** → **Close Measurement Windows** to close all currently open measurement windows.

6.1.6 The Calibration System

The calibration system is the same for online and offline experiments. The "Calibration Window" combo box lists all currently open calibration windows and offers the option of opening a new one. It is possible to have several elements in the same window, but only if they are of the same type, i.e. several tables can be in a table editor, several scalar elements can be in a numerical editor, etc. Assigning elements to calibration windows works like assigning elements to measurement windows. The calibration windows are described in chapter 6.2 on page 615.

To calibrate an element:

- In the "Elements" pane of the experimentation environment, select the element you want to calibrate.
- Choose **Elements** → **Calibrate**

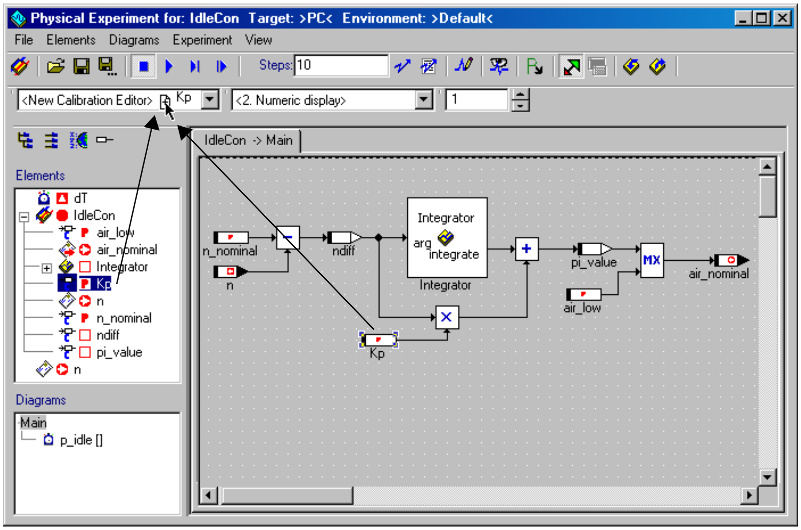
or

- double-click on the element.

  A data editor for the selected type of element is opened.

Or

- Drag the element from the "Elements" pane or the drawing area and drop it into the "Calibration Window" combo box.



If you drop the element on the entry `<New Calibration Window>`, a new window is opened, otherwise the element is added to the selected window. You can drag an element directly into an existing calibration window.

- Select **Experiment** → **Update Calibration Windows** to update the calibration windows content.
- Select **Experiment** → **Close Calibration Windows** to close all calibration windows.

6.1.7 Running Offline Experiments

After you have set up the experiment, you can start it. During offline experiments you can change the display options on all the measurement windows, you can open and close measurement windows, you can change the settings in the data and event generators, and you can change data values with the calibration system.

To start the offline experiment:

- Open the experimentation environment for the component or project you want to experiment with.
- Adjust the measurement rate in the "Measurement Rate" box.

  If the rate is set to 1, all measurement values will be displayed. If it is set to a higher value, only multiples of that value will be displayed, e.g. if the rate is set to 10, only every tenth measurement value is displayed.
- Click the **Expand / Collapse Window** button to hide the component display of the experimentation environment.

  Only the menu and button bars of the experimentation environment remain visible. Click the button again to make the component visible again.

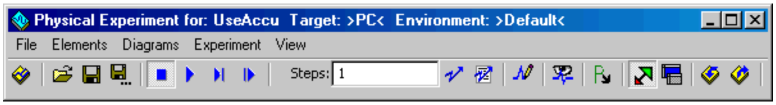
  In addition, the **Always on top** button is activated.
- Klick on the **Always on top** button, and the "Physical Experiment" window will always remain in the foreground.
- Select **Experiment** → **Start Experiment**

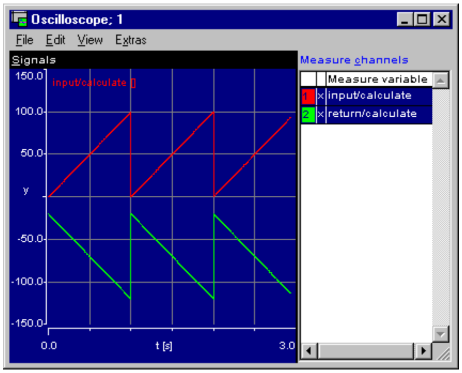
or

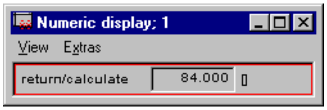
- click the **Start Offline Experiment** button.

  If you set the relevant options in the ASCET options window, "Experiment" node (cf. "Experiment Options" on page 59), variables and parameters are initialized each time this command is called.

The experiment is started and all the values you have set up in the measurement system are displayed in their respective windows.


Physical Experiment for: UseAccu Target: >PC< Environment: >Default<
File
Elements
Diagrams
Experiment
View
Steps: 1





- Make any necessary adjustments in the data generator, event generator, measurement system or calibration system.

To stop the offline experiment:

- Select **Experiment** → **Stop Experiment**.

or

- Click the **Stop Offline Experiment** button.

  The experiment stops but all settings remain active. The measurement data remains in the oscilloscope window and you can now analyze the data (see section "To analyze measurement data:" on page 665).
  Once you start the experiment again, the time axis is reset to 0, parameters and variables are initialized, if required (cf. page 59).

To leave the experiment environment:

- Click the **Exit to Component** button

or

- Select **File** → **Exit** to quit the experimentation environment and get back to the relevant editor for the component you have been simulating.

  You are asked if you want to save the current experiment environment.
- Activate the option **Remember my Decision** if you want to give the same answer to all questions of this type.
- In that case, the "Save Experiment" window no longer opens. You can revoke this setting in the ASCET options window, "Confirmation Dialogs" node (see "Options for Confirmation Dialogs" on page 45).
- Click **Yes** to confirm the saving.

  The experiment environment is stored under the suggested name. You return to the component editor.
- Click **No** to reject the saving.

  You return to the component editor without saving the experiment environment.
- Click **Cancel** to abort closing the experiment environment.

It is also possible to perform the experiment in single steps. At every step one event is activated in order of their priority.

To step through an experiment:

- Select **Experiment** → **Pause Experiment**

or

- click the **Pause Offline Experiment** button.

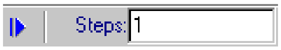
  The experiment is paused, i.e. no calculations are performed, but all settings remain active. When you start the experiment again, all data values remain intact.
- Select **Experiment** → **Step Experiment**

or

- click on the **Step Offline Experiment** button.

  The experiment is performed in single step mode.

  Repeat as necessary. You can also start stepping through an experiment when it is not running. In that case, its state automatically changes from *stopped* to *paused*.



Steps: 1

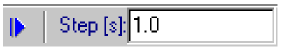
- You can adjust the number of steps to be performed in the "Steps" box before you step through the experiment.

  If, for instance, the number of steps is set to 5, five calculation steps are performed.

In the current version of ASCET, you can choose between three different types of step mode. An experiment either runs a number of steps, a number of seconds or until it reaches a breakpoint condition.

To switch to timed step mode:

- In the experimentation environment, choose **Experiment** → **Event Generator** → **Step Mode** → **Timed [s]** to activate the timed step mode.



Step [s]: 1.0

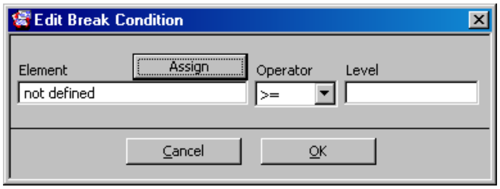
  The experimentation environment displays a text field for entering the number of seconds the experiment is to run. You can specify the number of seconds and then step through the experiment as described above.

The third possibility to step through an experiment using a breakpoint condition lets you specify a condition for an element in your component (e.g., number of revolutions is greater than 5,000). The experiment runs until the condition evaluates to true.

To set up a breakpoint condition:

- In the experimentation environment, choose **Experiment** → **Event Generator** → **Step Mode** → **Break At Condition** to enable the breakpoint step mode.

  The "Edit Break Condition" dialog box is displayed. In this dialog, you specify the condition up to which the experiment is to run.



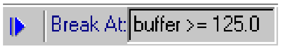
Element: not defined
Operator: >=

  This editor can be opened later with the **Experiment** → **Event Generator** → **Edit Break Condition** command.

- Click the **Assign** button to choose the name of the element you want to use in the condition.

  A selection prompter dialog box displays a list of elements.

- Select the desired element and click **OK** to use the element name.
- Edit the remainder of the condition by selecting an operator from the list and enter the corresponding value.
- Click **OK** to confirm.



Break At: buffer >= 125.0

  The experimentation environment displays a text field with the breakpoint condition. You can step through the experiment as described above.

You can view the implementation of the component you are experimenting with at any time during experimentation. You can also view the implementation of each element included in the component. It is of no importance whether the experiment is running or stopped.

To view the implementation:

- In the "Elements" list, select the element or component whose implementation you want to view.
- Select **Elements** → **Show Implementation**.

Note

This command is not available when you selected two or more elements.

The implementation editor for the selected object opens. Its usage is described in chapter 4.12 "Editing Implementations"; please note that the **OK** button is deactivated so that you cannot make any changes.

Components that are specified in C code provide additional facilities to display either debug information or error messages during experimentation. You can embed debug or error messages in your C code. Debug information is displayed in the target debug viewer that can be opened during experimentation. Error messages are printed to the ASCET monitor window.

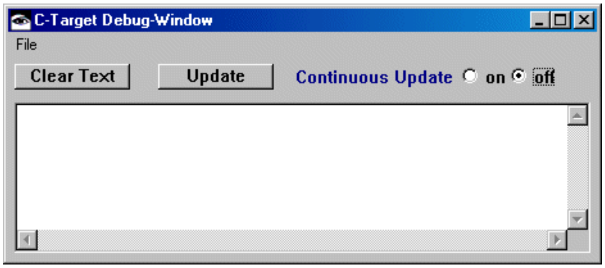
When editing C code, use the functions `asdWriteUserDebug()` and `asdWriteUserError()` to specify the information to be displayed. Both functions take an argument string which contains the message to be displayed. A typical statement could look like this:

```
asdWriteUserError("Overflow: \n Upper Limit exceeded.")
```

The string argument follows standard ANSI C rules, the example is printed in two lines.

To view debug information:

- Choose **Experiment** → **Open Target Debugger** to open the "C-Target Debug-Window".



Continuous Update: on [✓ off]

1. Automatic display

- In the "C-Target Debug-Window" window, activate the **on** option

or

- activate the **File** → **Continuous Update** menu function.

  Debugger information is displayed as soon as it is generated, and updated continuously.

2. Manual display

- Activate the **off** option

or

- deactivate the **File** → **Continuous Update** menu function to disable continuous update.

  With continuous update disabled, new debug information is only displayed upon request.

- For that purpose, click the **Update** button

or

- select **File** → **Update Text**.

3. Clear text window

- Click the **Clear Text** button

or

- select **File** → **Clear Text** to clear the debug window.

4. Save text window

- Select **File** → **Save as**.

  The file selection dialog opens.

- Choose a path and file name.
- Click **Save**.

  The content of the debugger window is written to a text file with the specified name and path.

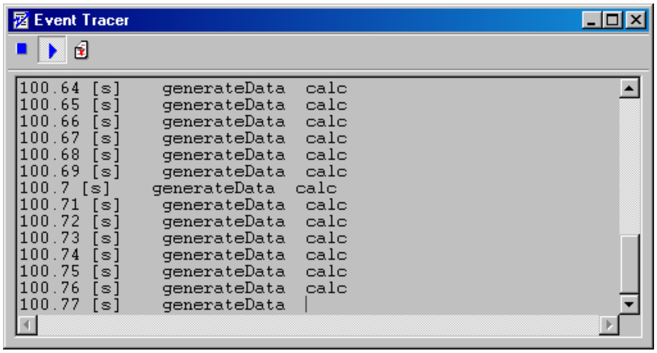
You can monitor which events are being triggered in which order with the event tracer. The event tracer shows the time in seconds for every point of time at which any events are triggered. For each point the events are listed in the order in which they have been triggered.

To monitor events with the event tracer:

- Set up the experimentation environment for the component you want to experiment with.
- Click on the **Open Event Tracer** button.

or

- Choose **Experiment** → **Event Tracer** to open the "Event Tracer" window.



```
100.64 [s]    generateData  calc
100.65 [s]    generateData  calc
100.66 [s]    generateData  calc
100.67 [s]    generateData  calc
100.68 [s]    generateData  calc
100.69 [s]    generateData  calc
100.7 [s]    generateData  calc
100.71 [s]    generateData  calc
100.72 [s]    generateData  calc
100.73 [s]    generateData  calc
100.74 [s]    generateData  calc
100.75 [s]    generateData  calc
100.76 [s]    generateData  calc
100.77 [s]    generateData  |
```

- In the "Event Tracer" window, click the **Start Tracer** button to start the event tracer.

- In the experimentation environment, start the experiment.
- To stop tracing events, click the **Stop Tracer** button.

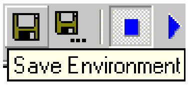
  You can start and stop tracing events while the experiment is running.
- Click the **Reset Tracer** button to clear the "Event Tracer" window.

6.1.8 Loading and Saving Environments

Experiments for complex models may consist of numerous measurement windows and measurement channels. It is therefore useful to be able to save the experimentation environment, so that the settings can be re-used between experimentation sessions. In ASCET you can save several such environments for every component.

When you save an environment, all settings in the data and event generators and the data logger are stored. Furthermore all open calibration editors and measurement windows are saved with their settings intact. Later you can load the environment, and the experimentation environment is restored. There is always a default experiment, which is loaded, if no other environments have been defined.

To save an environment:

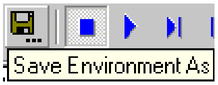


- Open the experimentation environment and make all necessary adjustments.
- Click on the **Save Environment** button

or

- select **File** → **Save Environment** to save the current environment.

To save an environment under a different name

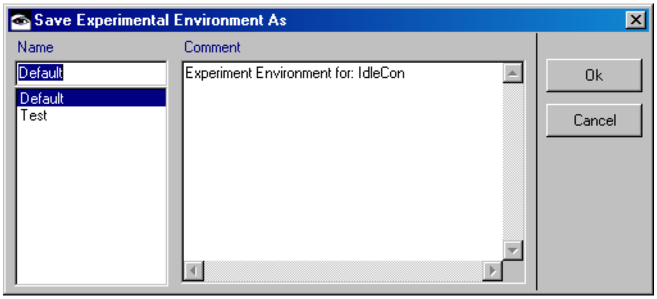


- Click on the **Save Environment As** button

or

- select **File** → **Save Environment As**.

  The "Save Environment As" dialog box opens.



Default

Default
Test

Experiment Environment for: IdleCon

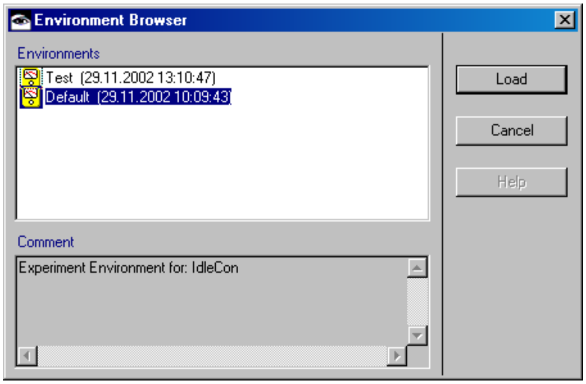
- Enter a name for the experiment.
- Enter a comment describing the experiment in the "Comment" pane.
- Click **OK**.

  The experiment is stored under the name you entered.

To load an environment:

- Open the experimentation environment.

  If there is only the default environment, it is loaded automatically on opening the experimentation environment. If you have saved more than one environment for the component, the Environment Browser opens with a list of all available environments.



Test (29.11.2002 13:10:47)
Default (29.11.2002 10:09:43)

Experiment Environment for: IdleCon

- Select the environment you want to load.
- Click **Load**.

  The experimentation environment opens with the selected environment.

It is possible to load environments, while the experimentation environment is already open. In that case, the currently open measurement and calibration windows remain open, but all other settings from the loaded environment become active, and all windows defined there are opened.

To switch environments:

- Click on the **Load Environment** button

or

- select **File** → **Load Environment**.

  The "Browse Environment" window opens.
- Select the environment you want to open.
- Click **Load**.

To export environments:

- Select **File** → **Export Experiment**.

  The "Export File" dialog box opens.
- Enter a path- and filename for the export file.

- Click on **Save**.

  The environments currently defined within the component are exported. Use the import mechanism of the Component Manager to import environments. The same rules apply to importing environments as to importing other data.

6.1.9 The Data Logger

With the data logger you can log the values of variables within a component or project during offline, or a project during online experimentation. The values are written to a file and can later be analyzed with a measurement data analysis application (e.g., the MDA). There are three logging modes:

- **Transient Sampling**

  For *Transient Sampling*, the code generation settings have to be adjusted. Code is then generated so that every time the value of a logged variable is changed, that change is automatically recorded. That way all changes in a value can be logged. Because the code is changed, transient sampling influences the runtime behavior of the model. The amount of data generated by the data logger increases, because a time stamp has to be generated for each logged value.

- **Periodic Sampling**

  Periodic sampling does not require any code modifications and therefore only minimally influences the runtime behavior of the model. Here, logging is triggered by a particular task, i.e. every time a selected task is triggered, the current value of all logged channels is recorded. This does not influence the runtime behavior of the selected task, as logging is performed only after the task is finished. If the value of a logged variable changes several times between subsequent logging operations, only the last change is recorded. No time stamp needs to be generated, because logging happens at pre-defined, fixed intervals.

  Logged data is stored on a ring buffer in the target and written to the PC-host after the recording is stopped, where it is written to another ring buffer. A ring buffer always stores a pre-defined number of values and, once that number is exceeded, overwrites the previously stored values on a first-in-first-out basis. Therefore, Data can be logged only for a limited time. To avoid this problem, you can activate *Continuous Polling*. With that, data is transferred to the PC-host at regular intervals during the logging operation. However, the communication with the host may affect the target processor, and gain and loss have to be weighed for each application.

- **Periodic to File**

  If a longer registering time is desired, the target, as a rule, cannot provide sufficient RAM for the recording. In that case, you can select *Periodic to File*. Here, too, logging is triggered by a particular task, but, different from Periodic Sampling, the data is transferred to the to the PC-host at regular intervals during the logging operation. Thus, data can be registered

For each of the three logging modes, data is written from the PC-host to a file once the logging operation is stopped (cf. page 607).

There are three limiting factors on the number of channels that can be logged and the rate at which data can be recorded:

- A portion of the *target RAM* is allocated to storing the logged data. The more RAM the target has, the more values can be logged.
- The *data transfer* between host and target influences the data logging. If, for instance, the target has little RAM, but the data can be transferred to the host very quickly, more values can be recorded.
- The logged values are stored on the host RAM (physical plus virtual), the more free RAM the PC has, the more data can be held.

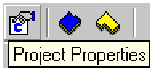
Before you can log data in transient sampling mode, you have to modify the code generation settings.

Note

*The **transient sampling** mode can only be activated from within a project.*

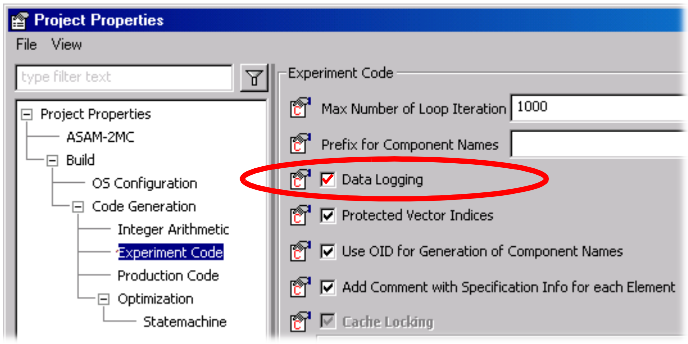
To prepare for transient sampling:

- Open the project you want to experiment with.
- In the project editor, click on the **Project Properties** button.



  The "Project Properties" window opens in the "Build" node.
- Open the "Experiment Code" node.

- Activate the **Data Logging** option to switch on data logging.



- Project Properties
  - ASAM-2MC
  - Build
    - OS Configuration
    - Code Generation
      - Integer Arithmetic
      - Experiment Code
      - Production Code
      - Optimization
        - Statemachine

Experiment Code

Max Number of Loop Iteration: 1000

Prefix for Component Names:

☑ Data Logging

☑ Protected Vector Indices

☑ Use OID for Generation of Component Names

☑ Add Comment with Specification Info for each Element

☑ Cache Locking

- Click **OK**.
- Generate the code for the project and open the desired experimentation environment.

If you want to log data in *Periodic Sampling* or *Periodic to File* mode, you do not have to modify the code generation settings. It is possible to generate code with or without data logging enabled. When you generate code with data logging enabled, the code will run more slowly, regardless of whether you are logging data or not.

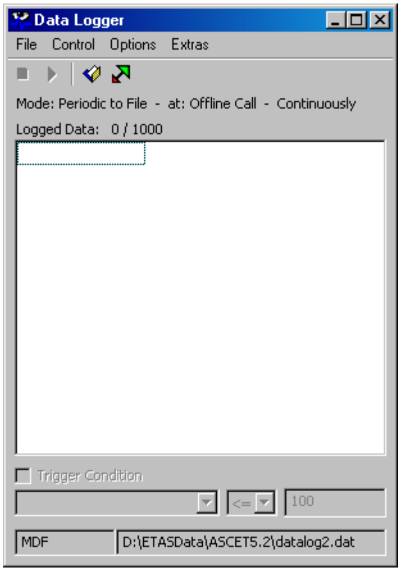
To open the data logger:

- Open the experimentation environment for the component or project.
- Click on the **ASCET Data Logger** button

or

- select **Experiment** → **Data Logger**.

  The "Data Logger" window opens.



Mode: Periodic to File - at: Offline Call - Continuously

Logged Data: 0 / 1000

MDF | D:\ETASData\ASCET5.2\datalog2.dat

This window contains the following elements:

- **File** menu
  - *Open (<CTRL>+<O>)*

    Opens a list (in the `*.lab` INCA format) with logged elements, see page 602.
  - *Save (<CTRL>+<S>)*

    Opens a list (in the INCA format `*.lab`) with elements currently logged in the data logger.
  - *Save As*

    Opens a list (in the INCA format `*.lab`) with logged elements under an arbitrary name.
  - *Import*

    Imports a list (in `*.csv` or SelectX – `*.cfg` – format) of logged elements.
  - *Exit*

    Closes the data logger.

- **Control** menu
  - *Enable Logging*

    Starts logging.
  - *Disable Logging*

    Stops logging.
- **Options** menu
  - *Logging Options*

    Opens the "Logging Options" window, see page 603.
  - *Change Filename*

    Changes the default name for the log file.
- **Extras** menu
  - *Convert MDF to FAMOS*

    Converts MDF data to FAMOS data, see page 609.
- Toolbar

■	Stops logging.
▶	Starts logging.
✎	Changes the default name for the log file.
⤢	Shows/hides the data logger pars below menu bar and toolbar.

- "Logged Data" field

  Lists the logged elements.
- Option **Trigger Condition**

  Activates/deactivates using a trigger condition (see page 606).
- Trigger definition

  Two combo boxes and an input field to define the trigger condition (see page 606).
- Status bar

  Status information for the data logger.

To set up the channels to be logged:

- In the experiment environment, select the desired elements from the "Elements" pane.

- Select **Elements** → **Log** to activate logging for the selected elements.

  When you have not yet opened the data logger, it opens now. The selected elements are listed in the "Logged Data" field.
- Alternatively, you can drag elements directly into the data logger.
- Select **Elements** → **Log All** to activate logging for all elements.

Note

The number of logged elements is limited by several factors, see page 598.

To use label lists:

If you close the data logger, the setup of registered elements gets lost. To ease setup for the next experiment, you can create a label list (`*.lab` format) for the elements currently in the data logger. This list can also be used for exchange with INCA, e.g. to make sure that the same elements are used in both tools.

- Open the data logger (page 599).
- Set up the channels you want to log (page 601).
- Select **File** → **Save** to save a list of the elements.

  If you save a list for the first time, you are asked for path and name of the file.
- Select **File** → **Save As** to save a list of the elements under an arbitrary name.
- Select **File** → **Open** to open an element list in the data logger.

  If the list contains elements not present in the current experiment, warnings are issued in the monitor window.

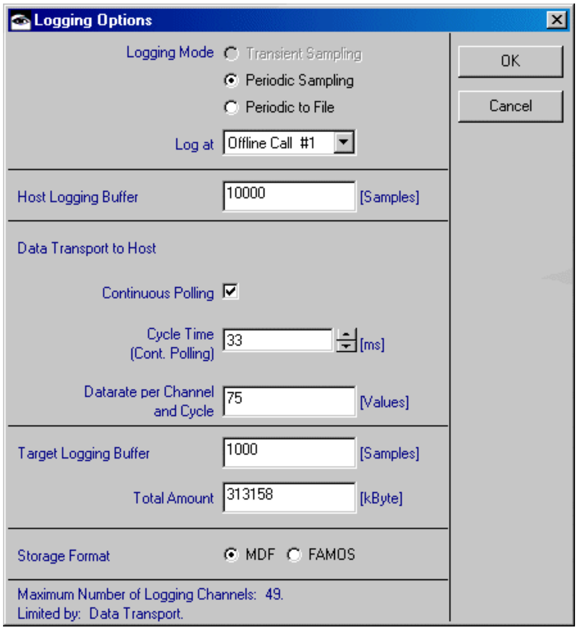
- Select **File** → **Import** to import a list in the `*.csv` (comma-separated values) or `*.cfg` (SelectX format).

  You can use these formats to create and manage label lists outside of ASCET.

To adjust the logging options:

1. Logging Mode

- In the data logger, select **Options** → **Logging Options** to open the "Logging Options" dialog window.



Logging Mode [ ] Transient Sampling [✓ Periodic Sampling] [ ] Periodic to File

Log at: Offline Call #1

Host Logging Buffer: 10000 [Samples]

Data Transport to Host

Continuous Polling: ✓

Cycle Time (Cont. Polling): 33 [ms]

Datarate per Channel and Cycle: 75 [Values]

Target Logging Buffer: 1000 [Samples]

Total Amount: 313158 [kByte]

Storage Format: [✓ MDF] [ ] FAMOS

Maximum Number of Logging Channels: 49.
Limited by: Data Transport.

- Click either the **Transient Sampling**, the **Periodic Sampling**, or the **Periodic to File** radio button.

  You can only select **Transient sampling** if data logging has been enabled in the code generation settings (cf. page 598).

- Select a task from the "Log At" combo box.

Note

The "Log at" combo box is invisible when **Transient Sampling** *has been selected.*

During *online* experiments, the combo box shows all tasks defined in the project. Logging then always occurs after the selected task has been executed.

During *offline* experiments, logging always happens after every event defined in the event generator; you do not have to select anything in that case.

- In the "Host Logging Buffer" field, adjust the channel length for the host logging buffer.

Note

The "Host Logging Buffer" field is deactivated when **Periodic to File** *has been selected.*

This setting determines how many values are stored in the ring buffer on the host-PC for each channel.

2. Data transfer to the host

- Tick or untick the **Continuous Polling** option.

Note

Continuous Polling *is activated automatically when* **Periodic to File** *has been selected.*

Without **Continuous Polling**, the data is stored on the target buffer and only written to the host at the end of a logging operation.

- In the "Cycle Time (Cont. Polling)" field, adjust the cycle time for continuous polling.

Note

The "Cycle Time (Cont. Polling)" field is not available when **Continuous Polling** *is deactivated.*

  If **Continuous Polling** is activated, the logging values that are stored on the target ring buffer are written to the host-PC at every interval defined in the "Cycle Time" field.

  When the cycle time cannot be kept, due to a long transportation period, the excess is indicated in the header of the data logger window. Optimization has to be done manually because computing speed, target load, and communication type have a strong influence.

- In the "Data Rate per Channel and Cycle" field, enter a data rate.

  The data rate setting determines, how many values per polling interval are read for every channel.

  If **Continuous Polling** is deactivated, the polling interval corresponds to the entire logging time; if **Continuous Polling** is activated, the polling interval corresponds to the cycle time entered in the "Cycle Time (Cont. Polling)" field.

3. Target register buffer

- In the "Target Logging Buffer" and "Total Amount" fields, adjust the target logging buffer settings.

  The target has a maximum amount of RAM that can be used as a buffer for logged data. If you want to use less RAM, you can adjust this setting. The channel length setting determines how many values for each channel are stored in the target ring buffer.

4. Output format

- Select a storage format for the log file by clicking either the **FAMOS** or **MDF** options.
- Click **OK**.

The settings in this dialog box determine the number of channels that can be logged simultaneously. The number is shown at the bottom of the dialog box and updated every time you change any settings. The current limiting factor is also shown here. If you enter a setting that exceeds any limits set by other settings or the available resources, an error message is shown, and the setting is not changed.

To log data:

- Start the experiment.
- In the data logger, click on the **Enable Logging** button

or

- select **Control** → **Enable Logging**.

  The logging of the selected elements is started.

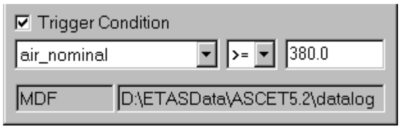
  When you select this command before starting the experiment in the experiment environment, logging is initiated. Data will be logged only after the experiment is started.

With that, data logging starts immediately when the experiment is running. The **Trigger Condition** option offers the possibility to define a condition for the start of data logging. To do so, proceed as follows:

To define a trigger condition:

- In the data logger, activate the **Trigger Condition** option.

  This option is only available when logging has stopped, and at least one element is listed in the "Logged Data" field.



☑ Trigger Condition

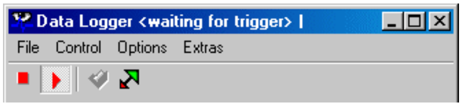
air_nominal | >= | 380.0

MDF | D:\ETASData\ASCET5.2\datalog

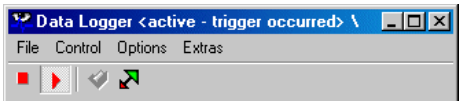
- From the left combo box, select one of the logged variables (e.g., `air_nominal` in the example).
- From the combo box in the middle, select a comparison operator.

  You can choose >= (greater or equal) or <= (smaller or equal).
- In the right field, enter a threshold value (e.g., `380`).

When you start the data logging now, the data logger postpones the actual logging until the condition is fulfilled. The data logger headline shows the status.



Once the condition is fulfilled, data logging starts, which is again shown in the headline. Data logging continues until it is switched off, even if the condition is no longer fulfilled.



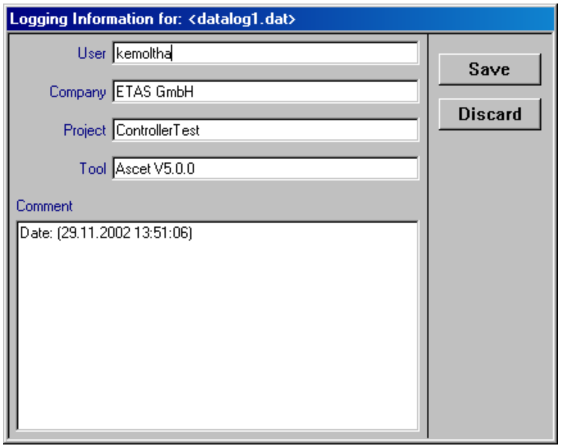
To stop data logging:

- Click on the **Disable Logging** button

or

- Select **Control** → **Disable Logging**.

  Data Logging is stopped. The "Logging Information" dialog box opens.



User kemoltha
Company ETAS GmbH
Project ControllerTest
Tool Ascet V5.0.0
Comment
Date: (29.11.2002 13:51:06)

- Fill in the information fields as needed.

  All information in this dialog is optional.

- Click the **Save** button to save the file,

or

- click **Discard** to discard the data.

The data is stored in a log file at the end of each logging operation. This file is named `datalog<n>.dat` by default, <n> being an integer number which is incremented each time data is saved, provided **Auto Increment DataLogger File Name** is activated in the ASCET options, "Datalogger" node (cf. page 58). If the option is deactivated, the log file is overwritten at the next logging. The log file is placed in the data directory of your ASCET installation, e.g., `ETAS-Data\Ascet5.2`.

To change the log file:

You can change the default name and path of the log file.

- Click on the **Change Filename** button

or

- select **Options** → **Change Filename**.

  The "File Selection" dialog window appears.

- Set the new path and filename.
- Click **OK** to adjust the log file.

  The filename at the bottom of the data logger window changes and the data is now written to the new file.

  When you set the respective option in the ASCET options window (cf. page 58), an integer number is added to the name you selected. This number is incremented each time data is saved.

To convert MDF data to FAMOS format:

- Select **Extras** → **Convert MDF to FAMOS**.

  The file selection dialog box opens for the input file.
- Select the MDF file you want to convert.

  The file selection dialog box opens for the FAMOS output file.
- Select a path- and filename for the FAMOS file.

  The MDF input file is read, converted to FAMOS, and written to the selected output file.

6.1.10 Experimentation Environment Options

Block Diagram Navigation

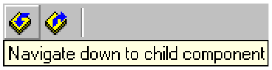
If a component or project includes other components, it is possible to view these components, without leaving the experimentation environment.

To navigate between block diagrams:

1. Navigate down

- Select the referenced component whose block diagram you want to view.
- Select **Elements** → **Show Component**

or



- click on the **Navigate down to child component** button

or

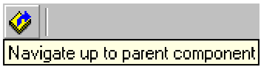
- in the component display, double-click on the occurrence of an included component.

  The selected component is shown in the experimentation environment.

2. Navigate up

- Select **View** → **Show Parent Component**

or



- click on the **Navigate up to parent component** button

or

- double-click on an empty place in the component display.

Note

Upward and downward navigation by double-click works only in block diagrams and state machines.

The parent component is displayed.

To switch between the diagrams of a component:

- Select the diagram you want to switch to in the "Diagrams" pane of the experimentation environment.
- Select **Diagrams** → **Show Diagram**.

  The selected diagram is displayed.

Display Options

To change the block diagram display:

- Select **View** → **Show Seq. Calls** to have the sequence calls of the current block diagram displayed.
- By default the sequence calls are not shown.

- Select **View** → **Hide Seq. Calls** to hide the sequence calls.
- Select **View** → **Redraw** to redraw the block diagram.

The four buttons above the Elements pane offer a variety of different views on the elements of the component.

To change the element display:

- Click the **Elements Hierarchical** button to switch from a flat list display to the default hierarchical view of elements.
- Click the **Elements Flat** button to switch to a list display.
- Click the **Exported** button to have only exported elements (i.e. messages and global elements) displayed in the "Elements" pane.
- Click the **Parameter Elements** button to have only parameter elements displayed.

6.1.11 Data Manipulation

During an experiment, the values of the elements are usually changed, e.g. to find the right parameter setting for a particular function. Once the right values have been found, they can be saved in the current data set of the component. For details see chapter "Data Sets" on page 466.

To read or write data from the current data set:

- Select the element whose value you want to write in the "Elements" pane.
- Select **Elements** → **Write Back Data** → **Selected Elements** or **Calibrated Elements**.

  A list of variables is displayed. If the command **Selected Elements** has been chosen, all elements currently selected in the "Elements" pane are shown in the list.
  If **Calibrated Elements** has been chosen, all elements that have been calibrated in the course of the experiment are shown. At first, all variables are selected.

- Deselect the variables you do not want to write back and click **OK**.

  The values of the selected elements are written to the current data set of the component. The old values of that data set is overwritten. The current data set is the one that was selected when the experiment was started.

- Select **Elements** → **Reinitialize** → **Variables** or **Parameters** or **Both** to read the data from the current data set.

  All variables and/or parameters are assigned the values from the current data set, the values in the experiment are overwritten. This command is not available while the experiment is running.

You can also write the parameter values to external files, or read them from external files.

Note

When you are experimenting with the ES1135 simulation controller, you can also file out the data of non-volatile variables to an external file.

To set up data exchange options:

- In the Component Manager, open the ASCET options window.
- In the "Data Exchange" node, adjust the options.
- Select **Elements** → **Data Exchange Options**.

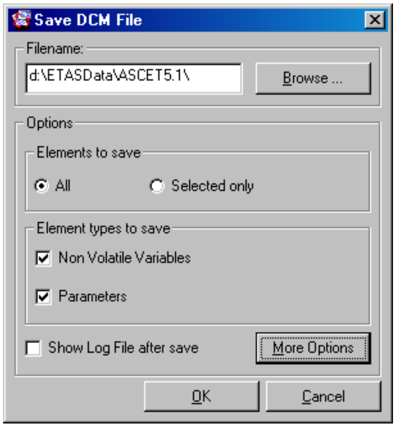
  The available options are described in section "Data Exchange Options" on page 65.

To write data to external files:

- In the "Elements" pane, select the parameters whose values you want to write to a file.

- Select **Elements** → **Save Data**.

  The "Save DCM File" window opens.



Filename: d:\ETASData\ASCET5.1\

Elements to save: [✓ All] Selected only

Element types to save:
- [✓] Non Volatile Variables
- [✓] Parameters

[ ] Show Log File after save

- Set the required options.
- Click on **OK**.

  The values of the selected parameters are written to the specified file. If required, the write log file is displayed.

The "Save DCM File" window offers the following options:

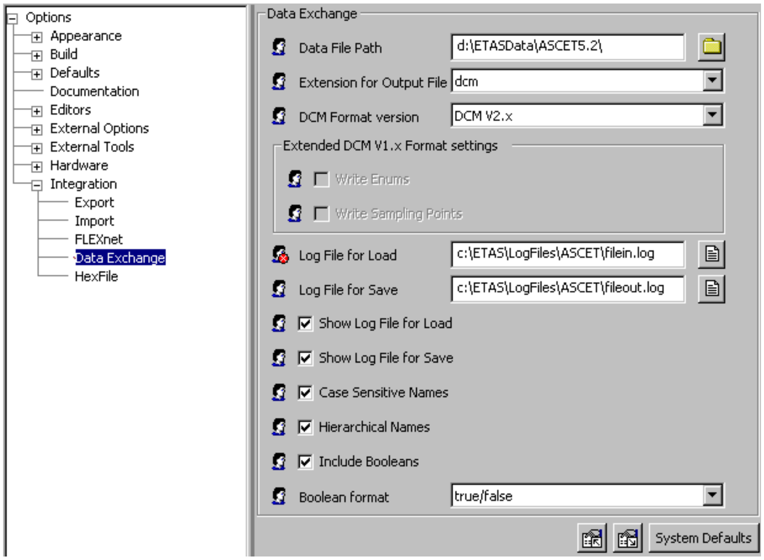
Note

The file name and at least one of the **Non Volatile Variables** *and* **Parameters** *options* ***must*** *be set explicitly. Otherwise, you cannot complete the procedure.*

- In the "Filename" field, enter path and name of the file either directly or via the **Browse** button.

  The file format is chosen according to the selection in the "Data Exchange" node of the ASCET options window (see page 65).
- Use the **All** and **Selected only** options to determine whether the data of all or selected elements are saved.
- Use the **Non Volatile Variables** and **Parameters** options to determine the element type(s) whose data will be saved.
- Use the **Show Log File after save** option to determine whether the log file is shown after the write process.

- Use the **More Options** button to open the ASCET options window in the "Data Exchange" node; the options are described in section "Data Exchange Options" on page 65.



Data Exchange

Data File Path: d:\ETASData\ASCET5.2\

Extension for Output File: dcm

DCM Format version: DCM V2.x

Extended DCM V1.x Format settings

☐ Write Enums

☐ Write Sampling Points

Log File for Load: c:\ETAS\LogFiles\ASCET\filein.log

Log File for Save: c:\ETAS\LogFiles\ASCET\fileout.log

☑ Show Log File for Load

☑ Show Log File for Save

☑ Case Sensitive Names

☑ Hierarchical Names

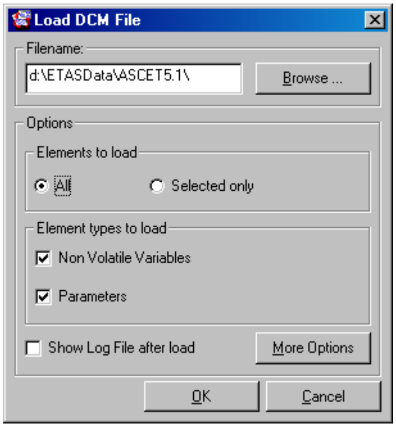
☑ Include Booleans

Boolean format: true/false

To read data from external files:

- Select **Elements** → **Load Data**.

  The "Load DCM File" window opens.



Load DCM File

Filename: d:\ETASData\ASCET5.1\

Browse ...

Options

Elements to load: (●) All ( ) Selected only

Element types to load:

[✓] Non Volatile Variables

[✓] Parameters

[ ] Show Log File after load

More Options

OK Cancel

  The optons are the same as in the "Save DCM File" window (cf. page 613). However, the **All** and **Selected only** options are irrelevant for reading data.

- Set the required options.
- Click on **OK**.

  The values of the selected parameters are written to the specified file. If required, the write log file is displayed.

6.2 Calibration Windows

You can use the calibration system to modify the values of the basic elements of the components you are experimenting with. You can alter the values when you set up the experiment, or while it is executing. It is also possible to assign initial values to elements during specification; however, modified values take priority over default settings in the calibration system.

If you assign a value to an element with the calibration system, this value remains until it is changed by a calculation within the component or overwritten by a value from the data generator. The data editors are the same as the ones used to specify components. They are described in the chapter "Editing Data" on page 458.

When specifying a component, you can assign an initial value to each element in your specification. All of these values—except constants—can be changed at a later stage. You can specify different data sets, i.e. sets of initialization values between which you can toggle, or you can change individual values during experimentation. This section describes the different editors for the various kinds of elements.

Usually a data editor is first called from within the component development environment, e.g. the block diagram editor, to assign a default value to an element. Then the editor can be opened again from within the experimentation environment, to calibrate the value of the element in the course of an experiment. Data editors can also be used to define data sets for components or projects. Data editors always work the same, regardless of which part of ASCET they were opened from.

How to open data editors from within the offline experimentation environment is described in section "The Calibration System" on page 584.

6.2.1 General Description of Menu Options

Not all menu functions are available in each editor.

- **Edit:**

Note

*The **Edit** menu is not available in the 3D graphical editor.*

- *Undo Last Change (<CTRL> + <U>)*
  Undoes the last change.
- *Redo Last Change d*
  Restores the last change.
- *Copy (<CTRL> + <C>)*
  Copies a calibration variable into the clipboard.
- *Paste (<CTRL> + <V>)*
  Pastes a calibration variable from the clipboard into the window.
- *Copy Entire Data Into Clipboard*
  The calibration variable (curve or map) active in the table editor is copied into the clipboard and can be inserted into other applications (e.g. Excel) from there and then processed further.
- *Select All Values (<CTRL> + <A>)*
  Selects all values.
- *Block Selection (<CTRL> + <B>)*
  Allows you to select several values on a curve.

- *Decrement* or *Decrement Value (*<CTRL> + <N>*)*
  Decrease by the preset value.
- *Increment* or *Increment Value (*<CTRL> + <M>*)*
  Increase by the preset value.
- *Add Offset*
  Adds one or several values.
- *Multiply By Factor*
  Multiplies by one or several values.
- *Fill With Values*
  Replaces one or several values by another value.
- *Decrement X Axis Point (*<CTRL> + <J>*)*
  Shifts x axis point to the left.
- *Decrement Arg (*<CTRL> + <J>*)*
  Shifts axis point in 2D graphical editor to the left.
- *Increment X Axis Point (*<CTRL> + <K>*)*
  Shifts x axis point to the right.
- *Increment Arg (*<CTRL> + <K>*)*
  Shifts axis point in 2D graphical editor to the left..
- *Add X Axis Point*
  Adds x axis point.
- *Remove X Axis Point*
  Removes the x axis point.
- *File In Data*
  Reads the data for an array or a table from a file
- *File Out Data*
  Writes the data from an array or a table to a file.

• **Axis:**

Note

*The **Axis** menu is only available when characteristic lines or maps are edited in the table editor.*

- *X Supporting Points Setup*
  Assigns values with a constant distance to the x supporting points.
- *Decrement X Axis Point (*<CTRL> + <J>*)*
  Decreases x axis point.
- *Increment X Axis Point (*<CTRL> + <K>*)*
  Increases the x axis point.

  - *Edit X Axis Point (*<CTRL> + <X>*)*
    Assigns a specific value to the x axis point.
  - *Add X Axis Point*
    Adds x axis point.
  - *Remove X Axis Point*
    Removes the x axis point.
  - *Y Supporting Points Setup..*
    Assigns values with a constant distance to the y supporting points.
  - *Decrement Y Axis Point (*<CTRL> + <R>*)*
    Decreases y axis point.
  - *Increment Y Axis Point (*<CTRL> + <T>*)*
    Increases the y axis point.
  - *Edit Y Axis Point (*<CTRL> + <Y>*)*
    Assigns a specific value to the y axis point.
  - *Add Y Axis Point*
    Adds y axis point.
  - *Remove Y Axis Point*
    Removes the y axis point.
- **View:**
  - *Larger font*
    Displays the calibrated value in a larger font size. If the command is reselected, the value returns to the original size.
  - *Display unit*
    Shows/hides the size unit.
  - *Reset Change Marks*
    Resets change marks.
  - *Show Process Point*
    Marks working point.
  - *Set Editor on Process Point (*<CTRL> + <W>*)*
    Scrolls to working point.
  - *Grid*
    Shows/hides display grid.
  - *xz-Viewpoint*
    xz representation.
  - *yz-Viewpoint*
    yz representation.

- *Show Key Help*
  Shows the most important keyboard commands in the bottom line of the window.

- **Extras:**
  - *Change Title*
    Changes the name of the measurement window highlighted.
  - *Optimize Size*
    Optimizes the size of the dialog window.
  - *Display Setup (*<CTRL> + <S>*)*
    Displays the setup window (the "Setup" dialog box depends on the editor type selected).
  - *Colors*
    Changes the color settings for the 1D graphical editor:
    Black & white → monochrome display,
    Default colors → default colors,
    Invert colors → inverts the current colors.
  - *Physical Representation (*<CTRL> + <P>*)*
    Represents the calibration variable(s) currently highlighted as a physical value.
  - *Hexadec. representation (*<CTRL> + <H>*)*
    Represents the calibration variable(s) currently highlighted in hexadecimal format.
  - *Decimal Representation (*<CTRL> + <Z>*)*
    Represents the calibration variable(s) currently highlighted in decimal format.
  - *Binary Representation (*<CTRL> + <R>*)*
    Represents the calibration variable(s) currently highlighted in binary format.
  - *Move Variable Into Window*
    Moves the calibration variable highlighted to another data editor Each variable can only be in one data editor at a time.
  - *Remove Variable*
    Removes the calibration variable highlighted from the data editor.
  - *About Variable (*<CTRL> + <I>*)*
    Displays a window containing information on the calibration variable highlighted.

– *Move* (only when a window contains several calibration variables)
Up → Moves the calibration variable highlighted up one position.
Down → Moves the calibration variable highlighted down one position.

6.2.2 Data Editors for Calibration Variables

Calibration is carried out directly in the display; this applies to all editors. In the numerical editors and table editors, calibration is carried out by changing the numeric values. Characteristic lines and maps can also be changed graphically by intuitively moving the break points.

When you have carried out a calibration, a red arrow is displayed next to the calibrated value. This indicates that the value of the corresponding calibration variable has been increased or decreased. This applies to all modified values, regardless of which editor you use.

6.2.3 Working with Calibration Windows

Every calibration window can display different variables; it is possible to move the variables between windows. When moving variables, each variable is deleted from one window and added to another. Each variable can only appear in one editor.

To move variables between calibration windows:

- Select one or more variables you want to move in a calibration window.
- Select **Extras** → **Move Variable Into Window**.

  The "Move variable" dialog box opens. It contains a list of all calibration windows corresponding to the data type.

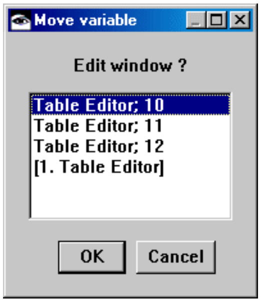


Table Editor; 10
Table Editor; 11
Table Editor; 12
[1. Table Editor]

- Select the window to which you want to move the marked variables.
- Click **OK**.

  The marked variables are moved to the selected window (new or existing) and removed from the original window. If the old window is empty after this action, it is closed.

To remove variables from a calibration window:

To remove variables from a calibration window, proceed as follows:

- Select one or more variables you want to remove by clicking on the relevant field.
- Select **Extras** → **Remove Variable**.

  The marked variables are removed from the window. If the window is empty after this action, it is closed.

To change the title of a calibration window:

To change the title of a calibration window, proceed as follows:

- Select **Extras** → **Change Title** in the calibration window whose title you want to change.
- Enter the new title in the input window.
- Click **OK**.

  The new title appears in the header of the calibration window.

To display information about variables:

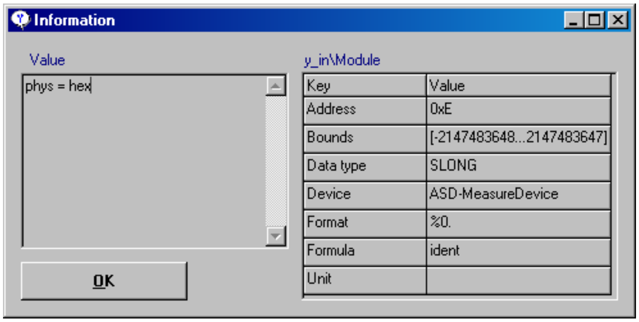
To display information about one or more variables, proceed as follows:

- In a calibration window, select one or more variables.
- In the same window, select **Extras** → **About Variable**

or

- press <CTRL> + <I>.

  An "Information" window displaying information about the respective variable opens for each selected variable.



Information

Value

phys = hex

y_in\Module

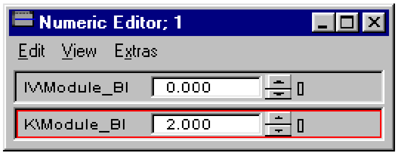
Key	Value
Address	0xE
Bounds	[-2147483648...2147483647]
Data type	SLONG
Device	ASD-MeasureDevice
Format	%0.
Formula	ident
Unit	

OK

- Click **OK** to close the "Information" window.

The different calibration windows are described in the following sections.

The Numerical Editor



Numeric Editor; 1

Edit View Extras

```
I\Module_BI   0.000
K\Module_BI   2.000
```

To set up a numerical editor:

- Select **Extras** → **Physical Representation**

or

- press <CTRL> + <P> to view the values as physical quantities.
- Select **Extras** → **Hexadec. Representation**

or

- press <CTRL> + <H> to view the values in hexadecimal format.
- Select **Extras** → **Binary Representation**

or

- press <CTRL> + <R> to view the values in binary format.
- Select **Extras** → **Decimal Representation**

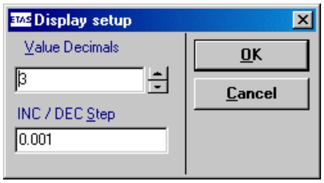
or

- press <CTRL> + <Z> to view the values in whole numbers.
- Select **Extras** → **Display Setup**

or

- press <CTRL> + <S> .

  The "Display setup" dialog box appears.



Value Decimals: 3

INC / DEC Step: 0.001

- Adjust the number of decimal places.

  This determines the number of decimal places with which the value is displayed.
- Adjust the value in the "INC/DEC Step" field.

  This value determines the step size for incrementation or decrementation.
- Click **OK**.

To edit a numeric value:

- Click inside the numerical display in the window.
- Edit the value and press <ENTER> to confirm your changes.

  Alternatively, you can adjust the value with the arrow keys displayed on the right of the value. This increments or decrements the value by the step size specified on setup. The new value is displayed immediately.

- Select **Edit** → **Undo Last Change**

or

- press <CTRL> + <U> to revert to the value before the last change.
- Select **Edit** → **Redo Last Change**

or

- press <CTRL> + <D> to reverse the undo operation.

Note

The last 10 changes of each variable are stored.

To edit several numerical values:

When several variables are contained in the same numerical editor window, you can change them together.

- Select one or more variables you want to change.
- Select **Edit** → **Increment**

or

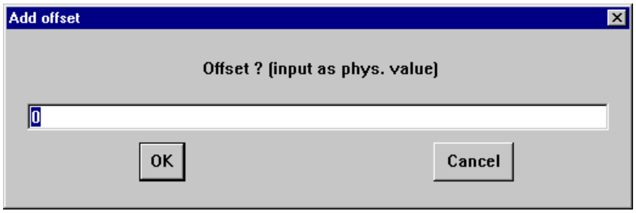
- press <CTRL> + <M> to increment all the highlighted values by the step size specified on setup.
- Select **Edit** → **Decrement**

or

- press <CTRL> + <N> to decrement all the highlighted values by the step size specified on setup.

- Select **Edit** → **Add Offset** to add a number to all the highlighted values.

  You are first prompted for an offset value, then that value is added to the value in the display.



Offset ? (input as phys. value)

0

- Select **Edit** → **Multiply By Factor** to multiply all the highlighted values with a factor.

  You are first prompted for a factor, then the multiplication is performed with that factor.
- Select **Edit** → **Fill With Values** to overwrite all the highlighted values.

  You are first prompted for a constant, which then replaces the value.

Note

*When you selected more than one value, you **cannot** undo the changes collectively.*

It is possible to exchange data with other applications via the Windows clipboard. These may be other calibration windows, but also other programs, like spreadsheets or databases.

To exchange data with other applications:

- Highlight the values you want to copy to the clipboard.

Note

Even though a numerical editor can contain several variables, you can only copy one variable at once.

- Select **Edit** → **Copy**

or

- press <CTRL> + <C> to copy all the values selected to the clipboard.
- To paste the data to another variable in a numerical editor, highlight that variable.
- To paste the data to a cell of a table, highlight the cell to which you want to paste it.

Note

Even if you select several cells, the value is copied only to the first one.

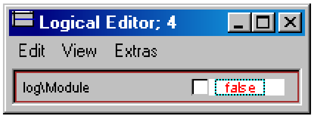
- Select **Edit** → **Paste**

or

- press <CTRL> + <V>.

  The data is pasted from the clipboard to the selected variable or cell, respectively.
- You can also paste the data into any other application.

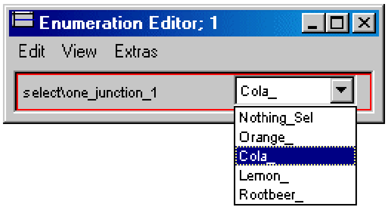
The Logical Editor



The menu functions correspond to the numerical editor menu functions of the same name.

To edit a logical value:

- Tick the option box to set the value to `True`.
- Untick the option box to set the value to `false`.



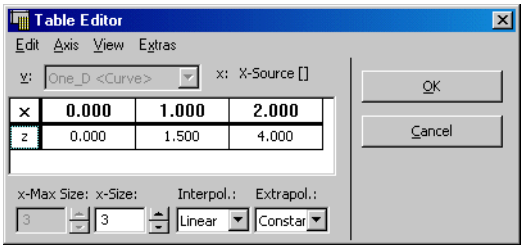
The menu functions correspond to the numerical editor menu functions of the same name.

To edit an enumeration:

The combo box contains all available enumerators.

- Select an enumerator from the combo box.

The Array Editor



x	0.000	1.000	2.000
z	0.000	1.500	4.000

You can have several arrays open at the same time in a single table editor window while experimenting with a component. In this case the tables currently open are shown in the "v:" combo box, and you can switch between them by selecting them from there. It is also possible to open tables of different kinds, e.g. arrays and 1-D tables, in the same editor window.

Note

Two dimensional arrays are also available in the form of matrixes.

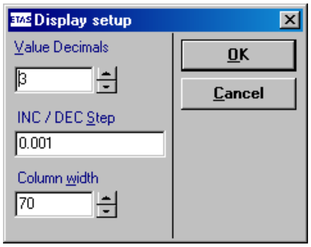
To set up the array editor:

- Select **Extras** → **Display Setup**

or

- press <CTRL> + <S>.

  The "Display Setup" dialog box appears.



Value Decimals: 3

INC / DEC Step: 0.001

Column width: 70

- Adjust the number of decimal places with which numbers are displayed in the array editor.
- Adjust the increment and decrement step size.
- Adjust the column width of the array editor.
- Click **OK** to close the "Display Setup numeric" dialog box.

To edit a single output value:

- Click on the value you want to edit.

  The value is highlighted for in-place editing.
- Enter the new value in the table cell.
- Press <ENTER>.

  The changed value is accepted and marked with an arrow.
- Select **Edit** → **Undo Last Change** to undo the last change.
- Select **Edit** → **Redo Last Change** to reverse the undo operation.
- Select **View** → **Reset Change Marks** to hide the arrows.

To edit several output values:

- Highlight the output value on the z axis you want to edit by moving the cursor over the value.

- Select **Edit** → **Select All Values**

or

- press <CTRL> + <A> to highlight all output values on the z axis.
- Select **Edit** → **Increment**

or

- press <CTRL> + <M> to increment all the values by the preset step size.
- Select **Edit** → **Decrement**

or

- press <CTRL> + <N> to decrement all the values by the preset stepsize.
- Select **Edit** → **Add Offset** to add a number to all the values highlighted.

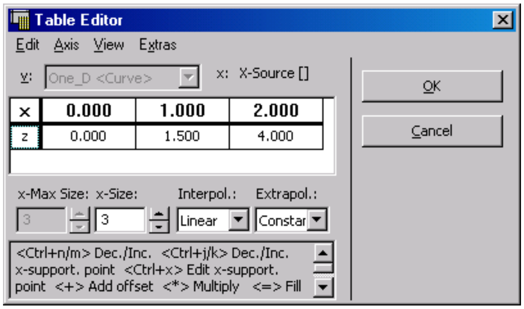
  When you choose this command, you are first prompted for a value which is then added to all the output values highlighted.
- Select **Edit** → **Multiply By Factor** to multiply all the values highlighted with a factor.

  When you choose this command, you are first prompted for a factor which is then used to perform the multiplication.
- Select **Edit** → **Fill With Value** to overwrite all the values highlighted.

  When you choose this command, you are first prompted for a constant which then replaces all the output values highlighted.

To change the display:

- Select **View** → **Show Key Help** to display the keyboard shortcuts for the array editor at the bottom of the window.



y: One_D <Curve> x: X-Source []

x	0.000	1.000	2.000
z	0.000	1.500	4.000

x-Max Size: 3 x-Size: 3 Interpol.: Linear Extrapol.: Constar

```
<Ctrl+n/m> Dec./Inc.  <Ctrl+j/k> Dec./Inc.
x-support. point  <Ctrl+x> Edit x-support.
point  <+> Add offset  <*> Multiply  <=> Fill
```

- Select **Extras** → **Optimize Size** to adjust the window size automatically when toggling between tables.

  This command is activated by default.

It is possible to exchange data with other applications via the Windows clipboard. These may be other array or table editor windows, but also other programs, like spreadsheets or databases.

To exchange data with other applications:

- Highlight the values you want to copy to the clipboard.

Or

- Select **Edit** → **Select All Values** to highlight all the values.
- Select **Edit** → **Copy** to copy the selected values to the clipboard.

Note

For tables, only the actual data values are copied, not the sample points.

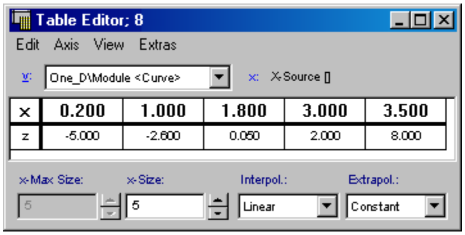
- Select **Edit** → **Copy Entire Data Into Clipboard** to copy all the data in the array to the clipboard.

- If you want to paste the data into another table or array editor, highlight the range of cells to which you want to paste (there must be the appropriate number of cells in the table).
- Select **Edit** → **Paste**.

  The data is pasted from the clipboard to the selected cells, respectively.

You can also paste the data into any other application. The data is stored in tab-delimited format in the standard Windows clipboard.

The 1-D Table Editor



v: One_D\Module <Curve>   x: X-Source []

x	0.200	1.000	1.800	3.000	3.500
z	-5.000	-2.600	0.050	2.000	8.000

x-Max Size: 5   x-Size: 5   Interpol.: Linear   Extrapol.: Constant

Several tables can be edited in the same table editor window, e.g. when you have selected several tables for calibration in the experimentation environment. In this case, all the available tables are shown in the "v:" combo box, and you can switch between them by selecting them from the said combo box.

The 1-D table editor works in the same way as the array editor. The only difference between the editors is that in the table editor the values on both axes can be edited. This section describes the differences between the two editors only; please refer to the previous section for all other information.

To set up the table editor:

- You can set up the table editor the same way as the array editor; see "To set up the array editor:" on page 627.

To use the sample point display:

With curve and map editors, you can display the sample point during a measurement.

- Select **View** → **Show Process Point** to activate the sample point display.

  The sample points enclosing the current value will be highlighted in the table (red border) during the next measurement.
- Select **View** → **Set Editor on Process Point**

or

- press <CTRL> + <W> to edit the sample pint next to the current value.

To edit existing sample points:

- Click on the sample point you want to change.

  A prompt box appears containing the current x value of the sample point.
- Enter the new value.

  The value has to be in between those of the previous and following sample points. If it is not, the change will not take effect.
- Click **OK**.

  When you have changed a value, a red arrow appears to the left of it.
- Select **Axis** → **Increment X Axis Point**

or

- press <CTRL< + <K> to increment the value of the axis point selected.
- Select **Axis** → **Decrement X Axis Point**

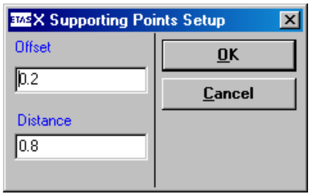
or

- press <CTRL< + <J> to decrement the value of the axis point selected.
- Select **View** → **Reset Change Marks** to hide the arrows.
- Select **Edit** → **Undo Last Change** to reverse your most recent action.
- Select **Edit** → **Redo Last Change** to cancel the undo operation.

To edit all sample points simultaneously:

- Select **Axis** → **X Supporting Points Setup**.

  The "X Supporting Points Setup" dialog box opens.



Offset: 0.2
Distance: 0.8

- Enter an offset and a distance into the relevant fields.
- Click **OK** to confirm your changes.

  The x axis points of the entire table are set up as specified. The first point gets the value specified as the offset, all other values are incremented by what is specified as the distance.

To insert new sample points:

- Select **Axis** → **Add X Axis Point** to insert a new sample point.

Note

If the table has already reached its maximum size, you cannot add any more sample points.

A prompt box appears.

- Enter the x value for the sample point.

  The new sample point will be inserted in the correct position within the table. It assumes the value of the sample point to its left.
- Click **OK**.
- Select **Axis** → **Remove X Axis Point** to remove the selected axis point.

To edit a single output value:

- Click on the value you want to edit.

  The value is highlighted for in-place editing.
- Enter the new value in the table cell.
- Press <ENTER>.

To edit several output values:

- Highlight the values you want to edit by moving the cursor over the value.

Or

- Select **Edit** → **Select All Values** to highlight all the output values on the z axis.
- Select **Edit** → **Increment**

or

- press <CTRL< + <M> to increment all the values highlighted.
- Select **Edit** → **Decrement**

or

- press <CTRL< + <N> to decrement all the values highlighted.
- Select **Edit** → **Add Offset** to add a number to all the values highlighted.

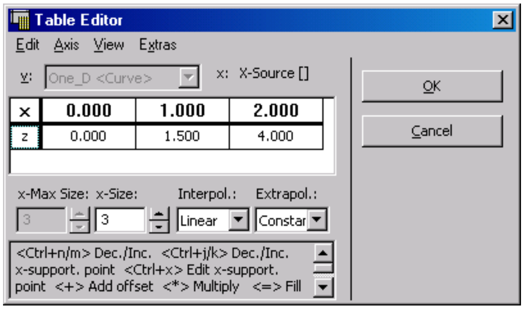
  When you choose this command, you are first prompted for an offset value which is then added to all the output values highlighted.
- Select **Edit** → **Multiply By Factor** to multiply all the values highlighted with a factor.

  When you choose this command, you are first prompted for a factor which is then used to perform the multiplication.
- Select **Edit** → **Fill With Values** to overwrite all the values highlighted.

  When you choose this command, you are first prompted for a constant which then replaces all the output values highlighted.

To change the display:

- Select **View** → **Show Key Help** to view the keyboard shortcuts for the 1-D table editor at the bottom of the window.



y: One_D <Curve>   x: X-Source []

x	0.000	1.000	2.000
z	0.000	1.500	4.000

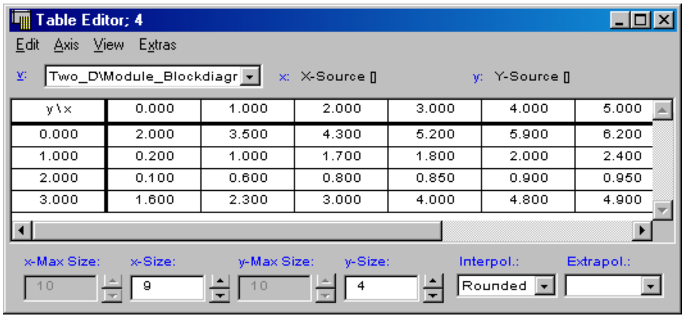
x-Max Size: 3   x-Size: 3   Interpol.: Linear   Extrapol.: Constar

<Ctrl+n/m> Dec./Inc. <Ctrl+j/k> Dec./Inc. x-support. point <Ctrl+x> Edit x-support. point <+> Add offset <*> Multiply <=> Fill

- Select **Extras** → **Optimize Size** to adjust the window size automatically when toggling between tables.

  This command is activated by default.

The 2-D Table Editor



y: Two_D\Module_Blockdiagr   x: X-Source []   y: Y-Source []

y \ x	0.000	1.000	2.000	3.000	4.000	5.000
0.000	2.000	3.500	4.300	5.200	5.900	6.200
1.000	0.200	1.000	1.700	1.800	2.000	2.400
2.000	0.100	0.600	0.800	0.850	0.900	0.950
3.000	1.600	2.300	3.000	4.000	4.800	4.900

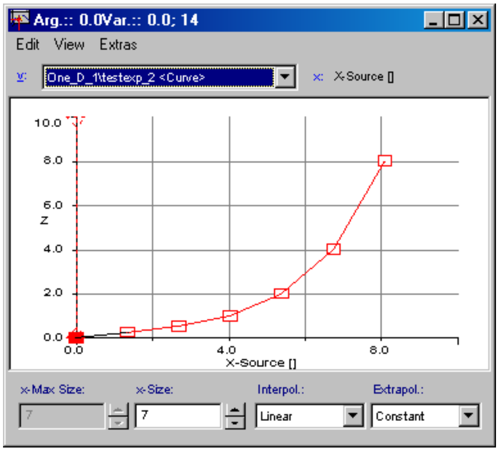
x-Max Size: 10   x-Size: 9   y-Max Size: 10   y-Size: 4   Interpol.: Rounded   Extrapol.:

The 2-D table editor works in the same way as the 1-D version described in the previous section. The only difference is that there are two dimensions of sample points instead of just one.

Editing the sample points and output values is the same as described in the previous section; the difference is that in the **Axis** menu there are separate commands for the x and y axis sample points. The 2-D Table also contains pairs of input boxes for the x and y values respectively.

Fixed and group tables are also available as 2-D tables and work in the same way as their 1-D equivalents.

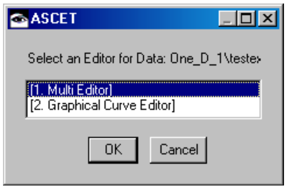
1-D Graphical Editor



Characteristic lines and maps can also be graphically displayed and edited. This section explains how to view and edit a 1-D table graphically.

Not all the graphical editor commands are dealt with here. The commands that are not explained here are identical to those in the 1-D table editor. The graphical editor is only available in the experimentation environment.

To launch the 1-D graphical editor:


ASCET
Select an Editor for Data: One_D_1\testex
[1. Multi Editor]
[2. Graphical Curve Editor]
OK
Cancel

- In the offline experimentation environment, right-click on the table you want to edit, and select **Calibrate** from the context menu.

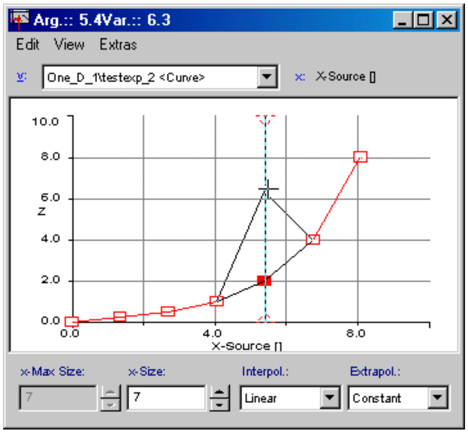
  A prompter is displayed with a list of all the editors available for the table selected.
- Select the `Graphical Curve Editor` entry.
- Click **OK** to open the editor.

  The graphical editor shows the x axis points as small squares connected by a red line.

To edit a table in the 1-D graphical editor:

- Click on the square representing the x axis point you want to edit.
- Drag the square up or down to change the value.

  The current value for the x axis point are displayed in the title bar of the editor window.



Arg.:: 5.4Var.:: 6.3

Edit View Extras

y: One_D_1\testexp_2 <Curve>   x: X-Source []

x-Max Size: 7   x-Size: 7   Interpol.: Linear   Extrapol.: Constant

- To change the value of the point on the x axis, move the vertical line cursor for the x-axis point selected.

  The x axis value is adjusted accordingly. You can move the cursor up to one of the neighboring sample points.

Or

- Select **Edit** → **Decrement X Axis Point** to decrement the selected x axis point.

Or

- Select **Edit** → **Increment X Axis Point** to increment the selected x axis point.
- Select **Edit** → **Block Selection**

or

- press <CTRL> + <B> to adjust more than one sample point.

  You can drag the mouse cursor around multiple sample points to select them. You can change the values of all these points together by moving them with the mouse.
  Adjusting multiple values only works along the z axis.

- Select **Edit** → **Block Selection** again to adjust one sample point only.

To add or remove sample points:

- Select **Edit** → **Remove X Axis Point** to remove the axis point selected.
- Select **Edit** → **Add X Axis Point** to insert a new x axis point.

Note

If the table has already reached its maximum size, you cannot add any more sample points.

  A prompt box appears.

- Enter the x value for the sample point.
- Click **OK**.

  The new sample point will be inserted in the correct position within the graphic. It assumes the value of the sample point to its left.

To change the graphical table editor display:

- Select **View** → **Grid** to hide the grid in the graphical display.

  This is a toggling menu choice, so you can choose it again to show the grid.

- Select **Extras** → **Colors** → **Black & white** to switch to monochrome display on the editor.
- Select **Extras** → **Colors** → **Default colors** to switch back to the default colors.

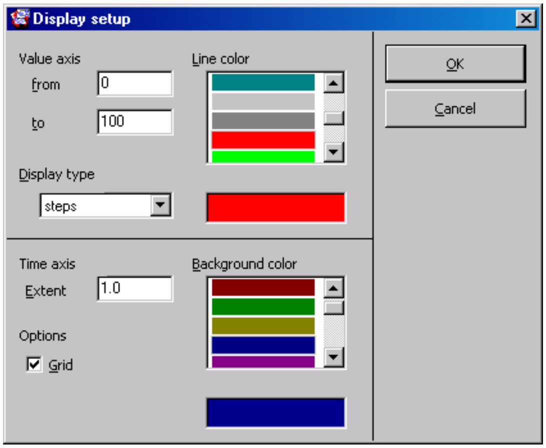
- Select **Extras** → **Colors** → **Invert colors** to invert the current colors.

Note

The colors used for the coordinate system and the labeling cannot be changed. The representation is usually in black, or if you select **Black & White** *or* **Invert colors***, in white.*

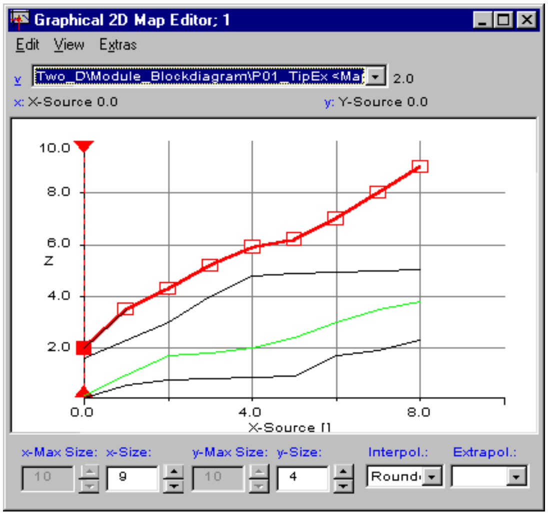
To set up the 1-D graphical editor:

- Select **Extras** → **Display Setup** to open the "Display setup" dialog box.



Value axis
from 0
to 100
Display type: steps
Time axis
Extent 1.0
Options
☑ Grid

- In the "Value axis" and "X-axis" fields, adjust the upper and lower limits of the value and the x axis.
- In the "Line color" field, select a color for the characteristic line.
- In the "Background color" field, select a background color.
- In the "INC/DEC Step" field, adjust the increment and decrement step size.
- Activate the **Grid** option if you want to display a grid.
- Click **OK** to confirm your changes.



With the 2-D graphical editor it is possible to edit 2-D tables graphically. This editor is similar to the 1-D graphical editor; it presents the 2-D table as a collection of 1-D tables that can be edited individually.

To start the 2-D graphical editor:

- In the offline experimentation environment, right-click on the table you want to edit and select **Calibrate** from the context menu.

  A prompter is displayed with a list of editors available for the table selected.
- Select the `Graphical 2D Map Editor` entry from the list.
- Click **OK** to open the editor.

To edit a table in the 2-D graphical editor:

- Select the line you want to edit by clicking on it.

  The selected line displays a rectangle for each sample point.

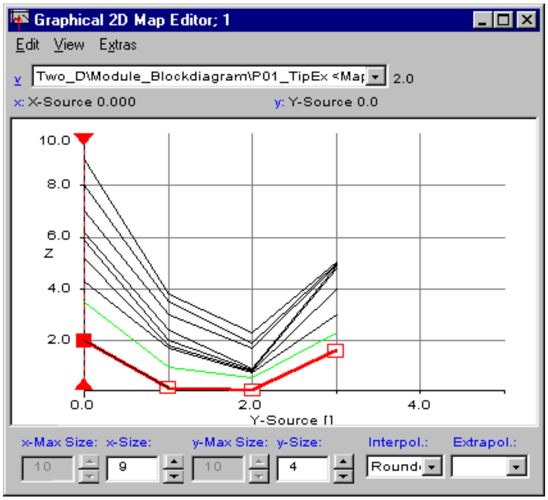
- Drag the sample point values for the selected line as in the 1-D graphical editor (see page 636).

  Here, the menu functions for incrementing/decrementing the sample points are named **Edit** → **Increment Arg** or **Edit** → **Decrement Arg**, respectively. According to the selected perspective (cf. page 641), they affect either the x or y axis.
- Switch to the previous or next line by pressing <CURSOR UP> or <CURSOR DOWN> or click to select another line.

The menu functions for adding or removing sample points are not available in the 2-D graphical editor.

To toggle the perspective of the 2-D editor:

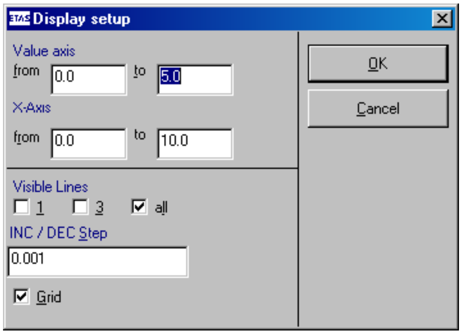
- Select **View** → **yz-Viewpoint** to swap the axes and the view the table from a different perspective.



- Select **View** → **xz-Viewpoint to** revert to the default perspective.

To change the 2-D editor display:

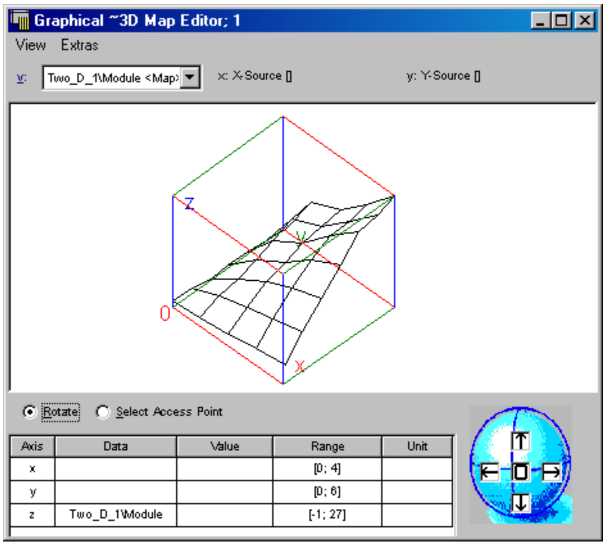
- Select **Extras** → **Display Setup** to open the "Display setup" dialog box.



Value axis from 0.0 to 5.0
X-Axis from 0.0 to 10.0
Visible Lines: [ ] 1 [ ] 3 [✓] all
INC / DEC Step: 0.001
[✓] Grid

- Adjust the ranges of the value axis and the x axis in the respective boxes.
- Check the line visibility option you want.

  If you enter 1, only the line currently selected is shown. However, you can still switch between lines with <CURSOR UP> and <CURSOR DOWN>.
- Uncheck the **Grid** option, if you do not want the grid to be shown.
- Adjust the increment and decrement step size.
- Click **OK** to confirm your changes.



v: Two_D_1\Module <Map>

x: X-Source []

y: Y-Source []

[✓ Rotate] Select Access Point

Axis	Data	Value	Range	Unit
x			[0; 4]	
y			[0; 6]	
z	Two_D_1\Module		[-1; 27]	

The 3-D table graphical editor shows a 2-D table as a three-dimensional graph. It is possible to rotate the graph in all directions and to measure and edit the values contained in the graph. The 3-D graphical editor is only available in the experimentation environment.

To start the 3-D graphical editor:

- In the offline experimentation environment, right-click on the table you want to edit and select **Calibrate** from the context menu.

  A prompter is displayed with a list of editors available for the table selected.
- Select the `Graphical 3D Map Editor` entry from the list:
- Click **OK** to open the editor.

  The graphical 3-D editor opens. The numerical table view is invisible while the graphical view is active.

The input area is displayed below the graphical representation. If the **Select Access Point** option is enabled, the values of the net point currently highlighted are displayed and can be calibrated by entering different values.

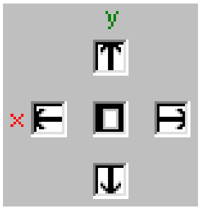
To highlight the net points in the 3-D Editor:

- Select the **Select Access Point** option located above the list.

or

- press the <s> key.

  A cross consisting of four arrows is displayed to the right of the list.



  Also, one net point is already highlighted: The point closest to the origin of the 3D coordinate system is highlighted by means of white connecting lines to its neighboring points.

- Now click on the direction arrows to move the highlight up and down or to the left and right

or

- use the ← ↑ ↓ → cursor keys.

  The x axis and y axis sample points, together with the corresponding output value, are shown in the "Value" column of the table. The table also shows the ranges of all three axes.

- Highlight the net point you want o edit.
- Enter the new values for X, Y and Z in the "value" column.

Note

For group and fixed characteristic maps, only the z axis values can be changed.

- Click on the square in the center of the arrows

or

- press the <0> (zero) key to reset the highlight to its original state.

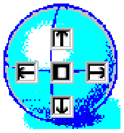
To highlight points in an awkward position, you may have to rotate the coordinate system.

To rotate the coordination system:

- Select the **Rotate** option located above the list

or

- press the <R> key.

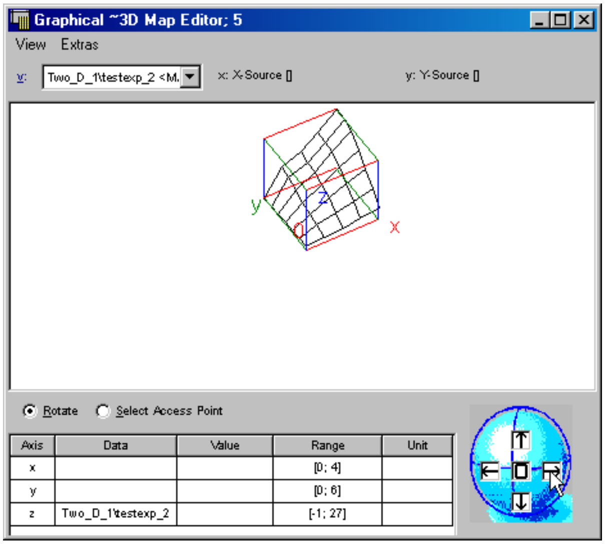


- Click on one of the arrows in the rotation control at the bottom right of the 3-D graphical editor window.

  You can rotate the display horizontally or vertically in either direction.
- Click the **0** button to revert to the original viewing angle.

- Click and hold down an arrow button to rotate the graph more quickly.

  The coordinate system is reduced to allow for faster rotation.

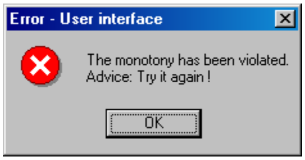


Axis	Data	Value	Range	Unit
x			[0; 4]	
y			[0; 6]	
z	Two_D_1\testexp_2		[-1; 27]	

6.2.4 Calibrating Sample Points in the Table Editor

Apart from the calibration commands described above for the **Edit** menu, the table editor offers additional commands for editing the sample points on the x axis and the y axis (**Axis** menu).

When you edit sample point values, the value you entered is checked for the required monotony. When the monotony is not kept, a dialog box will appear to inform you about this problem.



The monotony has been violated.
Advice: Try it again !

The sample point values of characteristic lines or maps can be edited in the x and y (y only for maps) table cells using the table editor. You cannot modify several sample points simultaneously, however. The corresponding commands are not available for arrays and matrices, even if these are displayed in the same editor window as characteristic lines/maps.

To calibrate sample points in the table editor:

The procedure described below applies both to characteristic lines and maps. For lines, however, the **Axis** menu contains only the commands for customizing the x axis.

- Using the mouse, directly click on the x axis value that you want to edit

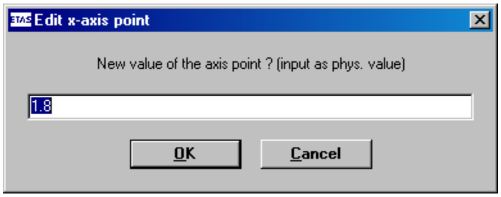
or

- press <CTRL> + <X>

or

- select **Axis** → **Edit X Axis Point**.

  The following dialog box appears:


Edit x-axis point

New value of the axis point ? (input as phys. value)

1.8

- Enter the value you want.
- Confirm with **OK**

or

- click on **Cancel** to cancel the operation.
- If you want to decrease the value by the preset amount:
  - Select **Axis** → **Decrement X Axis Point**

  or

  - press <CTRL> + <J>.
- If you want to increase the value by the preset amount:

  - Select **Axis** → **Increment X Axis Point**

  or

  - press <CTRL> + <K>.

- If you want to remove the sample point you selected, select **Axis** → **Remove X Axis Point**.
- If you want to add a sample point, select **Axis** → **Add X Axis Point**.

The procedure for editing the sample points on the y axis (maps only) is identical. Just use the corresponding commands for the y axis.

The sample points of fixed or group characteristic lines and maps, or distributions, cannot be edited. The **Axis** menu is deactivated (group/fixed characteristic lines/maps) or not present (distributions).

6.3 Measurement Windows

You select the appropriate measurement window as well as the appropriate display and measurement options depending on which measurement variables are to be used. The measurement windows can display measurements of the values for any element during an experiment in ASCET. A measurement channel is created for each element measured, and it is also possible to create several measurement channels for one element, e.g. if it is to be displayed in two different measurement windows. It is also possible to display measurement data without saving it, to display it and simultaneously start and stop recording or to record data without previously displaying it.

Note that offline experimentation is not real-time, whereas online experimentation always is.

6.3.1 Selecting Measurement Windows

In the experimentation environments, you have the choice of a number of measurement windows. The following measurement windows can be selected, sorted into experimentation environments.

- Oscilloscope
- Numerical display
- Vertical bar display
- Horizontal bar display
- Bit display
- Recorder

Section "The Measurement System" on page 581 describes how to open a measurement window.

6.3.2 General Description of Menu Options

The entries presented change depending on the window selected and on the experimentation environment.

- **Extras:**
  - *Change title*
    Changes the name of the measurement window highlighted.
  - *Message when out of bounds*
    Informs the user when a measurement value is above or below the measurement limit defined.
  - *Setup (<CTRL> + <S>)*
    Opens the setup window.
  - *Colors*
    Changes the color setting for the entire measurement window.
    Black & white – switches to monochrome display.
    Default colors – switches back to the default colors.
    Invert colors - inverts the current colors.
  - *Physical representation (<CTRL> + <P>)*
    Represents the measurement variable(s) currently highlighted as a physical value.
  - *Hexadec. representation (<CTRL> + <H>)*
    Represents the measurement variable(s) currently highlighted in hexadecimal format.
  - *Decimal representation (<CTRL> + <Z>)*
    Represents the measurement variable(s) highlighted in decimal format.
  - *Binary representation (<CTRL> + <R>)*
    Represents the measurement variables(s) in binary format.
  - *Copy variable to window*
    Copies the measurement variable highlighted to another measurement window.
  - *Move variable to window*
    Moves the measurement variable highlighted to another measurement window.
  - *Remove variable*
    Removes the measurement variable highlighted from the measure window.

  - *Change measure rate*
    Changes the sample rate (depends on the hardware configuration).
  - *About Variable* (<CTRL> + <I>)
    Displays a window containing information on the measurement variable highlighted.
  - *Attributes*
    Copy → copies the representation options for the measurement variable highlighted.
    Paste - assigns a representation option to the measurement variable highlighted. (Only possible if Copy performed previously.)
  - *Move* (only available with several measurement variables contained in one window)
    Up → Moves the measurement variable highlighted up one position.
    Down → Moves the measurement variable highlighted down one position.
    Left → Moves the measurement variable highlighted one position to the left (in the vertical bar display).
    Right → Moves the measurement variable highlighted one position to the right (in the vertical bar display).

- **File:**
  - *Print*
    Opens a dialog box for printing the oscilloscope or recorder window contents.
  - *Copy To Clipboard*
    Copies the oscilloscope or recorder window contents to the clipboard (screenshot) from which they can be pasted into any application.
  - *Save Selected Channels*
    Saves the measurement channel selected.
    MDF → Saves the measurement channel in MDF format.
    FAMOS → Saves the measurement channel in FAMOS format.
  - *Save All Channels*
    Saves all the measurement channels.
    MDF → Saves all the measurement channels in MDF format.
    FAMOS → Saves all the measurement channels in FAMOS format.
- **Edit:**
  - *Define trigger*
    Allows a trigger condition to be defined (only affects the display).

- *Activate trigger*
  Activates or deactivates the trigger.
- *Trigger manually*
  Sets off the trigger signal manually.
- *Autoscale (*<CTRL> + <A>*)*
  Adjusts the y axis scaling to a highlighted measuring channel.
- *Autoscale all channels*
  Adjusts the y axis scaling to all measuring channels.
- *Undo last scaling (*<CTRL> + <U>*)*
  Undoes the last scaling command.
- *Autodistribution*
  Distributes the channels highlighted to separate display areas.
- *Analyze measure data (*<CTRL> + <V>*)*
  Switches to analysis mode.
- *Analysis setup*
  Defines preferences for analysis mode (active only in analysis mode).
- *Analyze next point*
  Moves the measuring cursor to the next measurement point (active only in analysis mode).
- *Analyze previous point*
  Moves the measuring cursor to the previous measurement point (active only in analysis mode).

- **View:**
  - *Show selected channel*
    Shows/hides the measuring channel(s) highlighted.
  - *Grid*
    Allows you to define a background grid. The available options are "none", "dynamic" or "fixed" grid.
  - *Show measure channel lists*
    Shows/hides the list containing the measuring channels.
  - *Min/Max*
    Shows/hides the Min./Max. display in the measuring channel list.
  - *Rate*
    Shows/hides the sample rate in the measuring channel list.

- *Value at active cursor*
  Toggles on/off the display of the measuring channel values at the active cursor position in the measuring channel list (only in analysis mode).
- *Differences between cursors*
  Toggles on/off the display of the difference between the two cursor positions for the measurement variables in the measuring channel list (only in analysis mode).
- *Show key help*
  Show/hides the most important keyboard commands in the footer.
- *Show Setup*
  Shows/hides setting options in the footer.
- *Larger font*
  Switches the display to a large font size.

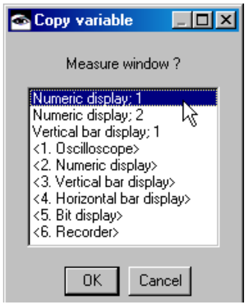
6.3.3 Working with Measurement Windows

Every measurement window can display the data from several measurement channels, and it is possible to copy and move measurement channels between windows. If a channel is moved, it is deleted from one window and added to another. If a channel is copied, a new measurement channel for the same element is created in a new window.

To copy channels between measurement windows:

- Select the channel or channels you want to copy by clicking on the relevant pane of the measurement window.
- Select **Extras** → **Copy variable to window**.

  The "Copy variable" dialog box opens. It contains a list of the available measurement windows.


Copy variable

Measure window ?

Numeric display; 1
Numeric display; 2
Vertical bar display; 1
<1. Oscilloscope>
<2. Numeric display>
<3. Vertical bar display>
<4. Horizontal bar display>
<5. Bit display>
<6. Recorder>

OK
Cancel

- Select the window you want to copy the variable to.
- Click **OK**.

  If you select a new window, it is opened with the copied channels. Otherwise, the channels are copied to the existing window.

To move channels between measurement windows:

- Select the channel or channels you want to move.
- Select **Extras** → **Move variable to window**.

  The "Move variable" dialog box opens. It contains a list of the available measurement windows.
- Select the window you want to copy the variable to.
- Click **OK**.

  The selected channels are moved into the selected window (either new or existing) and deleted from the old one. If the old window becomes empty, it is closed.

To delete channels from measurement windows:

To delete channels from a measurement window, proceed as follows.

- Select the channel or channels you want to move.
- Select **Extras** → **Remove variable**

or

- press <Del> (oscilloscope/recorder only).

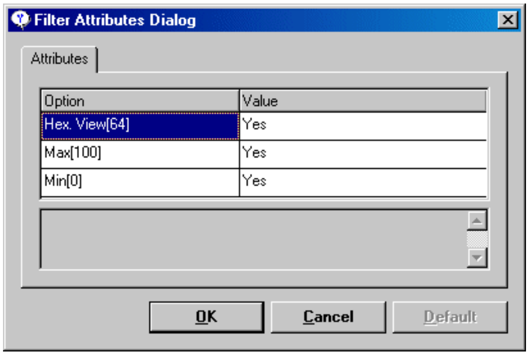
  The selected channels are removed from the window. If the old window becomes empty, it is closed.

Settings from one measurement window can be copied to another one, to facilitate setting up complex experiments. Only the settings applicable to a particular measurement window are copied, and you can choose which ones to copy.

To exchange attributes between measurement windows:

- In the measurement window from which you want to copy the settings, select **Extras** → **Attributes** → **Copy**.
- In the measurement window to which you want to copy the settings, select **Extras** → **Attributes** → **Paste**.

  The "Filter Attributes Dialog" dialog box opens. It lists all the attributes applicable to the new measurement window, together with the settings they had in the old one.



Option	Value
Hex. View[64]	Yes
Max[100]	Yes
Min[0]	Yes

- In the "Values" column, click on the value of an attribute.

  The field becomes a combo box.
- From the combo box, select `Yes` to select the attribute.
- From the combo box, select `No` to deselect the attribute.

  If you select `No`, that attribute is not applied to the new measurement window.
- Click **OK**.

  The attributes selected are copied to the new measurement window.

To change the display of a measurement window:

The menu functions described below are used to change the measurement windows display.

- In a measurement window, select oone or more channels whose display you want to change.
- Select **Extras** → **Physical representation** if you want to view the physical values.
- Select **Extras** → **Hexadec. representation** if you want a hexadecimal representation of the measurement values.

  This option is useful for experiments with fixed point code.

Note

The following two menu items are only available in the numerical display.

- Select **Extras** → **Change title** to change the title in the title line.
- Select **Extras** → **About variable** to open a dialog window with information about the element.
- Select **Extras** → **Message when out of bounds** to display a message window each time a measured value falls outside the monitoring limits set up for the channel.

  This command is not available in the oscilloscope or recorder.

To change a measurement window title:

To change the title of a measurement window, proceed as follows:

- In the measurement window whose title you want to change, select **Extras** → **Change Title**.
- Enter the new title in the input window.

- Click **OK**.

  The new title appears in the header of the measurement window.

To display information about variables:

To display information about one or more channels, proceed as follows:

- In a measurement window, select one or more channels.
- In the same window, select **Extras** → **About Variable**

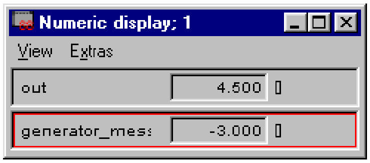
or

- press <CTRL> + <I>.
- An "Information" window displaying information about the respective variable opens for each selected channel.
- Click **OK** to close the "Information" window.

6.3.4 The Different Measurement Windows

Numerical Display

The numerical display shows the values of the measurement channels it is launched for, in either decimal, binary or hexadecimal representation.



```
out              4.500
generator_mes:  -3.000
```

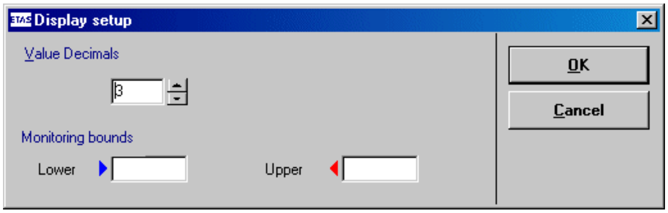
To set up a numerical display:

- Select the numerical display you want to set up by clicking on it.

  You can select more than one representation within the same window by clicking on the selection and holding down the <CTRL> key.

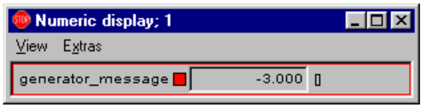
- Select **Extras** → **Setup**.

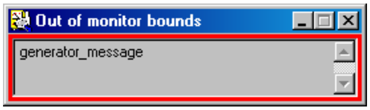
  The following dialog box appears.



- In the "Value Decimals" field, enter the number of decimal places with which the measured value is displayed.
- The fields under "Monitoring bounds" allow you to define one or two monitoring limits.
- Clock **OK** to close the setup window.

  When a value falls below or exceeds the monitoring limits, a red warning lamp lights up in the title bar. In addition to this, a blue square to the left of the display indicates that the value fell below the lower limit, and a red square indicates that the value exceeded the upper limit.
  In addition, a message window opens that closes automatically once the measurement value falls again between the limits.





To change the display options for a numerical display:

- Select **Extras** → **<Format> representation** to view the measurement values as described on page 655.

- Select **Extras** → **Move** → **Up** or **Down** to move the measurement channel selected in the numerical display window up or down.

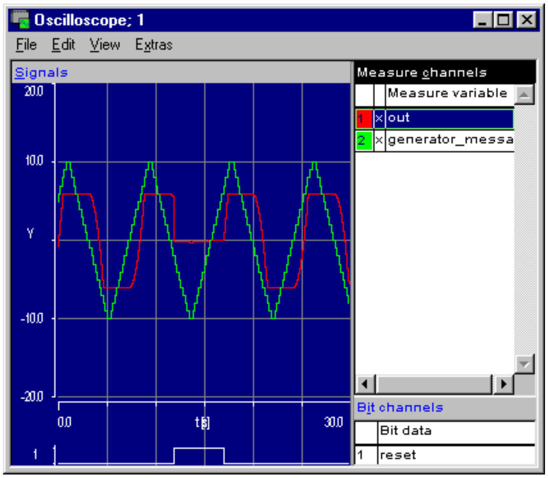
  This command is only available if there is more than one channel in the window.
- In the offline experimentation environment, select **View** → **Larger font** to view the measurement in a larger font size.

Oscilloscope

The oscilloscope provides a set of highly flexible display options, which are similar to those of a real oscilloscope. It is often a good idea to open several oscilloscope windows at the same time, as too many channels can clutter up the display.

When you use the oscilloscope, you usually start by setting up the display options, i.e. define the way the various values are shown in the window. During the experiment you can perform various operations on the data displayed, e.g. store it, analyze it, define triggers etc.

The "Signals" pane shows the curves for the numerical values being measured. Underneath is the "Bit Channels" display area which shows the logical values being measured. To the right of the "Signals" pane is the "Measure Channels" field, where the names of the channels are displayed, together with various user-definable pieces of information about them. The following illustration shows the oscilloscope in the offline experiment environment.



		Measure variable
1	x	out
2	x	generator_messa

	Bit data
1	reset

Changing the Oscilloscope Display:

The following options affect the measurement channels selected. You can select one channel, all channels or any combination of channels. The options can be set both before and during a measurement.

To set up measurement channels:

- In the "Measure Channels" pane, select the channels you want to set up.

  You can select more than one channel by clicking on them and holding down the <CTRL> key.

Note

Bit channels cannot be set up.

- Select **Extras** → **Setup**

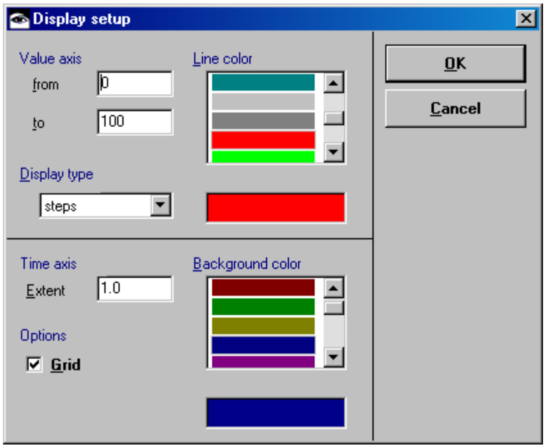
or

- double-click on the channel you want

or

- press the <S> key.

  The "Display setup " dialog box opens.



Value axis
from 0
to 100
Display type steps
Time axis
Extent 1.0
Options
☑ Grid

- In the "from" and "to" fields, adjust the lower and upper limits.

  The values you enter here form the upper and lower limits of the y axis in the oscilloscope display. The default range is 0–100.
- In the "Line Color" field, select a color for the channel.

Note

This option os available only when you selected a single channel in the "Measure channels" pane.

  The channel you selected is now displayed in that color. You can display several values with the same color, but that may make the display less clear. By default, every value is assigned a different color.
- From the "Display type" combo box, select the line style.

  If you select `steps`, a step is drawn between each two measurement values, if you select `line`, they are connected by a straight line.
- Click **OK** to accept the settings.

The "Display Setup" dialog box contains some options that do not affect individual channels, but are applied to the entire oscilloscope window.

To set up the oscilloscope window:

- Open the "Display Setup" dialog box as described above.
- Adjust the value in the "Time Axis" box.

  The value you enter here determines the time slot that is displayed by the oscilloscope, e.g. if you set the value to 1, the oscilloscope will show the output of one second, if you set it to 0.5 of half a second etc. The default value is 1 second.

Note

Keep in mind that, during offline experiments, these values do not correspond to the real time the calculations take.

- From the "Background Color" field, select a background color.
- Tick or untick the **Grid** option button to display the axis grid in the display area or hide it.
- Click **OK**.

  The settings become active. When you changed the time slot, the curves already displayed in the display area are deleted..

To change the color settings for the oscilloscope:

- Select **Extras** → **Colors** → **Black & white** to select a monochrome display.
- Select **Extras** → **Colors** → **Default colors** to restore the default colors.
- Select **Extras** → **Colors** → **Invert colors** to invert the current color settings.

The following options are used to scale the channels; the measurement can be running or paused. Automatic scaling uses the current window content to compute the adjustment. You can select one channel, all channels or any combination of channels.

To scale measurement channels:

- Select **Edit** → **Autoscale.**
- The y axis is adjusted for each channel to fit the values of the particular channel.
- Select **Edit** → **Autoscale all channels** to perform auto-scaling on all the channels simultaneously.

  This has the same effect as performing the operation for each channel individually.

- Select **Edit** → **Autodistribution** to distribute the selected channels to separate display areas.

  Using the current window content, the channels are sccaled in a way that they do not overlap in the display area.

- Select **Edit** → **Undo last scaling**

or

- press <CTRL> + <U> to undo the last scaling operation.

Note

When **Autoscale all channels** *was used without selecting all channels, the scaling of the unselected channels is* ***not*** *undone with this command.*

To show/hide measurement channels:

You can hide one or more channels without deleting them from the oscilloscope (or recorder). Proceed as follows.

- Select one or more channels.
- Select **View** → **Show selected channel**

or

- Press the <X> key.

  The selected channels are hidden or shown in the display area, but they remain visible in the "measure channels" list.

To show/hide lists:

To show or hide the "Measure channels" and "Bit channels" lists, proceed as follows.

- Select **View** → **Show measure channel lists**

or

- Press the <L> key.

  The "Measure channels" and "Bit channels" lists are hidden.

- Repeat the step to display the lists again.

Note

The lists cannot be hidden or shown individually.

To set up the lists (acquisition mode):

To set up the "Measure channels" and "Bit channels" lists, proceed as follows.

- Select **View** → **Min/Max.**
- The "Min..Max" column is added to the "Measure channels" and "Bit channels" lists. It contains the minimum and maximum values for each channel.
- Select **View** → **Rate**.

  The "Rate" column is added to the "Measure channels" and "Bit channels" lists. It contains the sample rate used for each channel.

To display keyboard commands:

To display the keyboard commands, proceed as follows.

- Select **View** → **Show key help**.

  The keyboard commands are shown at the bottom of the oscilloscope window.

  Only those keyboard shortcuts that are relevant in a particular context are displayed, e.g. the shortcuts for analysis mode are not displayed in acquisition mode.
- Repeat the command to switch off the display.

To display the setup:

To display the current setup for a channel, proceed as follows.

- Select a channel.
- Select **View** → **Show Setup**.

  The current settings for the axes at the bottom of the oscilloscope window.

  When you select another channel while the display is switched on, the settings of the new channel are shown.

- Repeat the command to switch off the display.

By default the oscilloscope is in *acquisition mode*, i.e. it displays the data generated by the experiment. There is also an *analysis mode*, to analyze your data further. Analysis mode is only available when the experiment is stopped or paused.

To start the analysis mode:

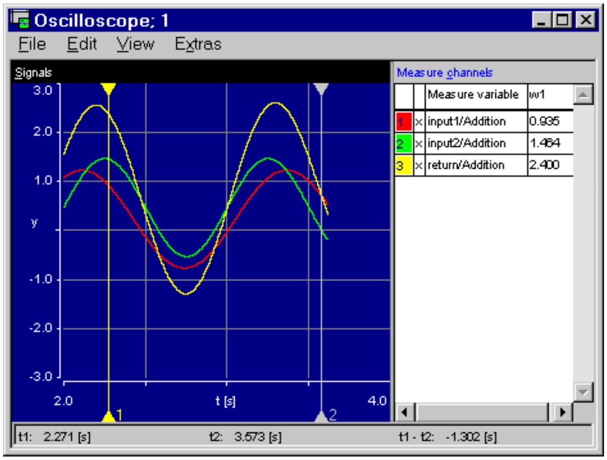
To start the analysis mode, proceed as follows.

- Select **Edit** → **Analyse measure data**

or

- press <CTRL> + <V>.

  Two vertical lines appear in the display area. The originally left line is number 1, the other is number 2. These lines are the measurement cursors, which define two points at which data is measured. The active measurement cursor is yellow, the other one is gray.
  At the bottom of the oscilloscope window, the time values, i.e. the position of both cursors on the x axis, as well as their difference, are shown.



		Measure variable	w1
1 🟥	x	input1/Addition	0.935
2 🟩	x	input2/Addition	1.464
3 🟨	x	return/Addition	2.400

t1: 2.271 [s]   t2: 3.573 [s]   t1 - t2: -1.302 [s]

To set up the lists (analysis mode):

To set up the "Measure channels" and "Bit channels" lists, proceed as follows.

- Select **View** → **Value at active cursor**.

  The "wx" column appears in the "Measure channels" and "Bit channels" lists. It displays the value of the active measurement cursor. "x" can be 1 or 2, depending on which cursor is active.

- Select **View** → **Difference between cursors**.

  The "wx - wy" column is added. It displays the difference between the value at the active cursor and the value at the inactive cursor. "x" is the number of the active cursor, "y" is the number of the inactivbe cursor.

- Resize the window or the display area so that all the information is visible.
- If necessary, drag the vertical separation lines to asjust the column width so that you can read all the information.

To analyze measurement data:

To analyze the data measured in the oscilloscope, proceed as follows.

- Start the analysis mode (cf. page 664).
- Select the measurement cursor you want to move.

  The measurement cursor becomes active. The "wx" and "wx - wy" columns in the "Measure channels" and "Bit channels" lists are updated. "x" is the number of the active cursor, "y" is the number of the inactive cursor.

- Drag the measurement cursor to the desired position

or

- use the cursor keys to move the cursor.

  The current measurement values at the cursor position are displayed in the "Measure channels" list. The x axis values are displayed at the bottom of the oscilloscope window.

- If necessary, resize the oscilloscoope window.

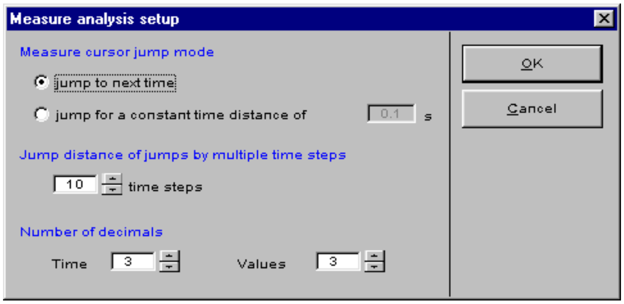
- If necessary, drag the vertical separation lines to asjust the column width so that you can read all the information.

There are various options that affect the way measurements can be analyzed. These options are available only when the oscilloscope is in analysis mode.

To set up analysis:

- Select **Edit** → **Analysis setup.**

  The "Measure analysis setup" dialog window.


Measure analysis setup

Measure cursor jump mode
[✓ jump to next time]
jump for a constant time distance of 0.1 s

Jump distance of jumps by multiple time steps
10 time steps

Number of decimals
Time 3 Values 3

- In the "measure cursor jump mode" field, select the jump mode that is appropriate for you.
  - **Jump to next time** means that the cursor moves continuously from one value to the next.
  - **Jump for a constant time distance of ...** means that the cursor moves stepwise. You can specify the step size.

    If, for instance, you set the value to 0.1, each move of the cursor will result in a jump by 0.1 seconds on the x-axis.
- In the "time steps" box below "Jump distance of jumps by multiple time steps", enter a multiplier for the value in the **s** box.

  If, for instance, the step value is set to 0.1, and the multiplier is set to 3, the actual jump distance on the x-axis for each cursor movement will be 0.3 seconds.

- In the "Time" box below "Number of Decimals", enter the number of decimals for the time values.
- In the "Values" box, enter the number of decimals for tne measurement values.
- Click **OK**.

  The new settings are accepted. They will become active only when you move a measurement cursor.

If a trigger condition is active, the oscilloscope will not display any data until the trigger condition is met. Once the condition has been met, the oscilloscope will display the values as normal. Triggering only relates to the display of values in the oscilloscope, it does not influence the calculations in the experiment in any way.

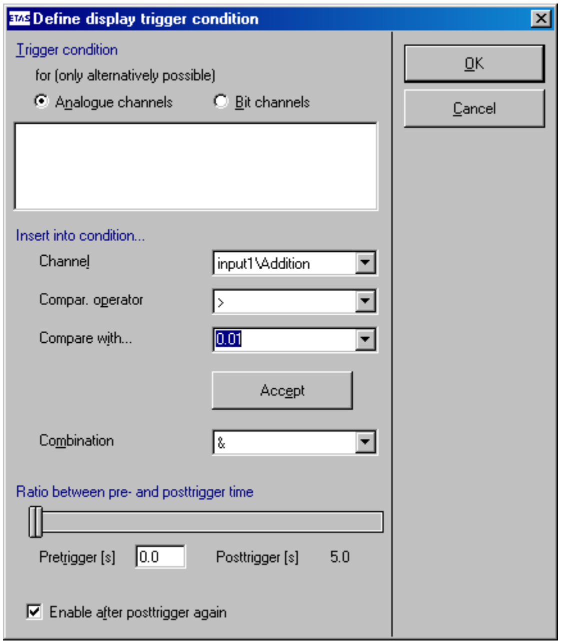
You can only define a trigger if the experiment is stopped or paused.

To define simple trigger events:

To define a simple trigger, proceed as follows.

- Select **Edit** → **Define trigger**.

  The "Define display trigger condition" dialog opens.



Define display trigger condition

Trigger condition
for (only alternatively possible)
[✓ Analogue channels] Bit channels

Insert into condition...
Channel: input1\Addition
Compar. operator: >
Compare with...: 0.01

Combination: &

Ratio between pre- and posttrigger time
Pretrigger [s]: 0.0 Posttrigger [s]: 5.0

[✓] Enable after posttrigger again

- Select a trigger mode by clicking the **Analogue channels** or the **Bit channel** option.

  Trigger conditions can only be defined for either analog or bit channels.

- In the "Channel" combo box, select a channel.

- In the "Compar. operator" combo box, select a comparison operator.

  The following operators are available: == (equal), > (greaterTo defin than), < (less than), `exceeds` and `false`.

- In the "Compare with" combo box, select the channel you want to compare with the first,

or

- enter a number, `true` or `false`.

Note

You must use a point as decimal point. Using a comma will produce an error message.

- Click on **Accept**.

  The condition you devined in the previous steps is written to the text field.

- As an alternative to the preseecing three steps, you can enter the condition directly in the text field.
- Click **OK** to close the window.

  The trigger condition is now defined. It is automatically, and will be used when you start the experiment.

In the example shown here the condition is triggered when the value of the `input1/addition` channel is greater than `0.01`. It is also possible to compare channels by selecting a channel in the "Compare With" combo box.

To define a multipart trigger:

The condition can have several parts which are combined by a logical `and` or a logical `or`. To define a multipart trigger, proceed as follows:

- Open the "Define display trigger condition" window.
- Select a trigger mode by clicking the **Analogue channels** or the **Bit channel** option.

  Trigger conditions can only be defined for either analog or bit channels.

- Define the first part of the trigger condition.

  The procedure is the same as for simple triggers (cf. page 668).

- In the "Combination" combo box, select & (and) or | (or) as operation.

- Define the second part of the trigger condition.

  Repeat the necessary steps if you want to add more parts.
- Click on **OK**.

  The trigger condition is now defined. It is automatically, and will be used when you start the experiment.

Since the oscilloscope buffers the values even if the trigger condition is not yet fulfilled, values can be displayed afterwards for a definable time (pre-trigger time) before the trigger event happens. The post-trigger time determines the length of time for which the values are shown after the trigger event has happened. The pre- and post-trigger time is collected along the extent of the time axis on the oscilloscope window. If, for instance, the time axis extent of the oscilloscope window is set to 2 seconds, and the ratio between pre-trigger and post-trigger time is 0.4/0.6, the pre-trigger time is 0.8 seconds, and the post-trigger time is 1.2 seconds.

To set pre-/post-trigger time:

Pre- and post-trigger time can only be set in acquisition mode, with the experiment paused or stopped. To set the times, proceed as follows.

- Open the "Define display trigger condition" window.
- Define a trigger condition.
- Adjust the ratio between pre- and post-trigger time by moving the **Ratio between pre- and posttrigger time** slider

or

- in the "Pretrigger [s]" field below the slider, enter the pre-trigger time in seconds.

  The pre-trigger time cannot be longer than the length of the time axis in the oscilloscope window. The post-trigger time is calculated automatically; it is the difference between the length of the time axis and the pre-trigger time.
- Activate the **Enable after posttrigger again** option if triggering is to restart after the post-trigger time has expired.

- Click **OK.**

  The settings are accepted.

Once you have defined a trigger condition, it is activated automatically. It will be used as soon as the experiment is started the next time. You can also activate, deactivate and actuate the trigger manually.

To activate/deactivate a trigger:

You can activate or deactivate a trigger only in acquisition mode, when the experiment is stopped. To activate a trigger, proceed as follows.

- In the oscilloscope window, select the **Edit** menu.

  The trigger is activated wen the **Activate trigger** menu option is checked. It will be used when you start the measurement.

- Select the **Activate trigger** menu option to deactivate the active trigger condition.
- Select **Edit** → **Activate trigger** again to re-activate the trigger.

To actuate the trigger manually:

Manual actuation of a trigger is only possible in acquisition mode while the measurement is running. To activate a trigger manually, proceed as follows.

- Select **Edit** → **Trigger manually**.

  This has the same effect as if the trigger condition was met.

You can copy the oscilloscope window to the clipboard and paste it into other applications. It is also possible to print out the oscilloscope window.

To copy the oscilloscope window.

- Stop the experiment.
- Highlight the oscilloscope window you want to copy to the clipboard.
- Select **File** → **Copy to Clipboard**.

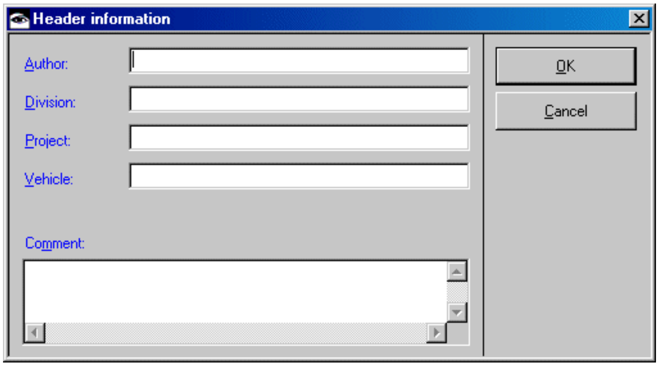
  This copies the oscilloscope window to the clipboard.

To print the oscilloscope output:

To print the display area Signals in the old experiment environment, proceed as follows:

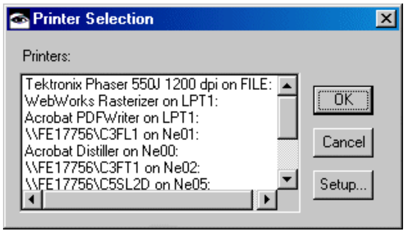
- Stop the experiment.
- Select **File** → **Print**.

  The "Header information" window opens.



- Enter the necessary information about author, department, project, and vehicle in the respective fields.
- In the "Comment" field, enter additional comments.
- Click **OK**.

  Your settings are accepted, and the "Printer Selection" window opens.



Tektronix Phaser 550J 1200 dpi on FILE:
WebWorks Rasterizer on LPT1:
Acrobat PDFWriter on LPT1:
\\FE17756\C3FL1 on Ne01:
Acrobat Distiller on Ne00:
\\FE17756\C3FT1 on Ne02:
\\FE17756\C5SL2D on Ne05:

- Select a printer and click **OK**.

  The "Signals" pane is printed on the printer you selected.
  The "Measure channels" and "Bit channels" lists are not printed.

You can store the measurement data in the file system in either MDF or FAMOS format. The file format is standardized and can be read by other ETAS products.

Due to the multitasking behavior of Windows, the data set saved in the file may not always be complete during an offline experiment. Use the data logging feature (see section "The Data Logger" on page 597) in an online experiment for complete accuracy.

To save the oscilloscope data in a file:

- Stop the experiment.
- Select one or more channels you want to save.
- Select **File** → **Save Selected Channels** → **<format>**.

  You can select either MDF or FAMOS.

Or

- Select **File** → **Save All Channels** → **<format>** to save the contents of all channels in the oscilloscope.

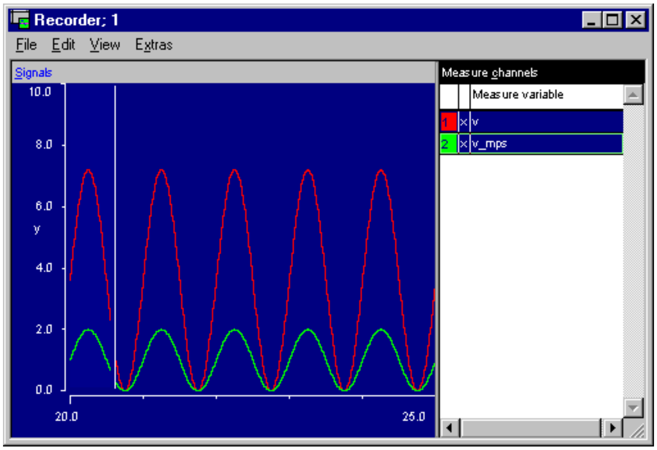
  The "Store measure data" dialog box opens.
- Select a path and file name for the output file.
- Click **Open**.

  The data of the curves in the current oscilloscope window are stored in the selected format and file.

Recorder

The recorder works in a similar fashion to the oscilloscope. The major difference is that the display area is not updated between passes. The oscilloscope is described in the previous section. In an oscilloscope window, the content of the display area is deleted every time the output curves reach the right-hand side of the window.

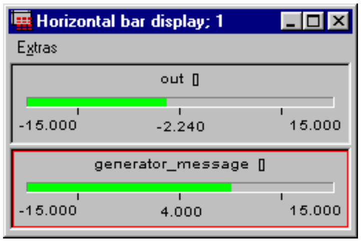
This is not the case in a recorder window; the output curves remain on the display until they are overwritten explicitly. A cursor in the form of a white vertical line indicates up to which point the curves have been overwritten; the curves to the right of the line are those from the previous pass.

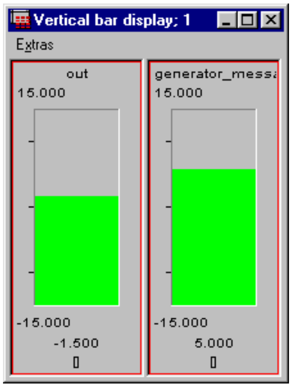


The only other difference between the recorder and the oscilloscope is that you cannot display a grid in the display area of a recorder window, because that grid would have to move with the cursor. The display options, measurement data analysis and trigger feature work in the same way as in the oscilloscope. However, only those parts of the curves belonging to the current pass are available for analysis; the parts to the right of the white line cannot be analyzed.

Horizontal and Vertical Bar Display

The bar display represents measured data as colored bars. You can define the upper and lower limits of the display as you wish. If the measured value falls below the lower limit specified, the display remains blank. The measured value is also displayed numerically in the middle below the bar. The vertical bar display can only be selected in the offline experimentation environment.

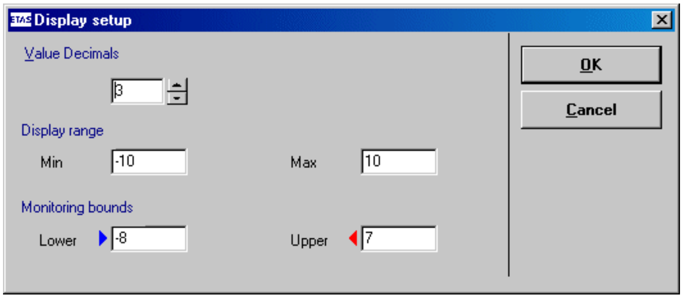




To set up the bar display:

- Select the bar display you want to set up.
- You can also select several bar displays simultaneously by clicking on them while holding down the <CTRL> key.
- Select **Extras** → **Setup**.

  The following dialog box is displayed:



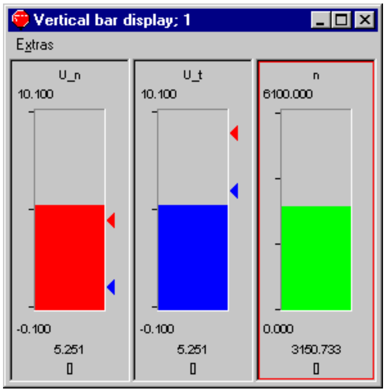
Value Decimals: 3

Min: -10 Max: 10

Lower: -8 Upper: 7

- In the "Value Decimals" field, select the number of decimals for displaying the numeric value.
- In the "Min" and "Max" fields, set the display range.
- In the "Lower" and "Upper" fields, define one or two monitoring limits.

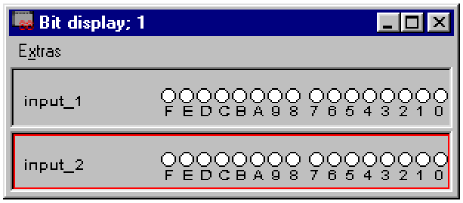
  In the bar display, the monitoring limits are marked by colored triangles. If the measured value falls below the lower limit, the bar color changes to blue. If the measured value exceeds the upper limit, the bar color changes to red. If the measured values stay within the two limits, the bar color is green..



- Click **OK** to accept the settings.

Bit Display

The bit display represents measured values as a bit array. It is especially useful for representing binary channels as well as for a quick reading of dual digits when measurement is paused.


Bit display; 1
Extras
input_1
F E D C B A 9 8 7 6 5 4 3 2 1 0
input_2
F E D C B A 9 8 7 6 5 4 3 2 1 0

The bit display cannot be set up. You can copy (cf. page page 652), move (cf. page 653) and delete (cf. page 653) channels, exchange attributes with other measurement windows (cf. page 654), change the window title (cf. page 655) and display channel information (cf. page 656).

To move channels in the bit display window:

- Select **Extras** → **Move** → **Up** to move the selected channel up in the bit display window .

- Select **Extras** → **Move** → **Down** to move the selected channel down in the bit display window.

  This command is only available if there is more than one channel in the window.

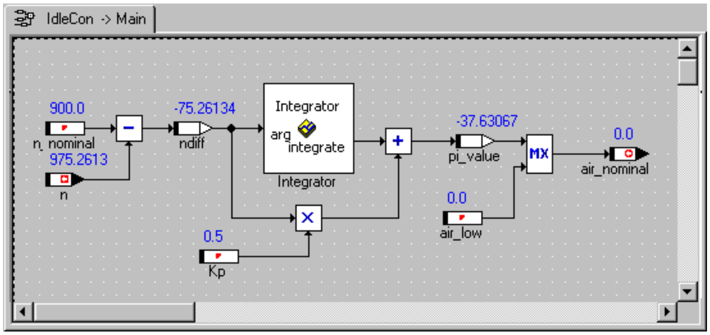
Monitor

Monitors are a simple way of viewing numerical and logical values inside a block diagram. They can be particularly useful for keeping track of the way different values influence each other in complicated diagrams with many elements.

To monitor an element:

- In the drawing area of the offline experimentation environment, right click on the element you want to monitor.
- Select **Monitor** from the context menu.

  When you start the experiment, the value is displayed above the element in the "Physical Experiment" window.



- Select the **Monitor** command again to deassign the monitor.
- Select **View** → **Monitor All** to assign monitors to all elements.
- Select **View** → **Delete Monitors** to delete all the monitors in the diagram.

7 Automatic Documentation

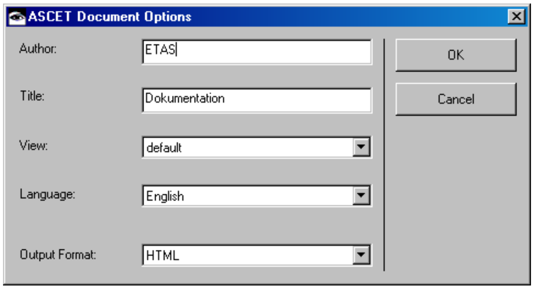
ASCET can automatically generate a documentation file for each folder or database item. The documentation file contains information about the documented item, e.g. the interface and specification diagrams, plus any notes the user adds. Documentation can be generated in either RTF, HTML, ASCII or Postscript format, and then printed or used within other documents.

7.1 Generating Documentation

Before the documentation can be generated, the folders or items that should be documented must be added to the contents of the documentation. It is possible to generate documentation for one or more entire folders, or just for one or more database items.

To set the options for documentation generation:

- In the Component Manager, select **Tools** → **Documentation** → **Options** to open the "ASCET Document Options" dialog window.


ASCET Document Options

Author: ETAS

Title: Dokumentation

View: default

Language: English

Output Format: HTML

- Type your name into the "Author" field and type a title for the generated document into the "Title" field.
- Select the view of the documentation file from the "View" combo box (see chapter 7.3 for details).
- Select the language for the documentation file from the "Language" combo box.

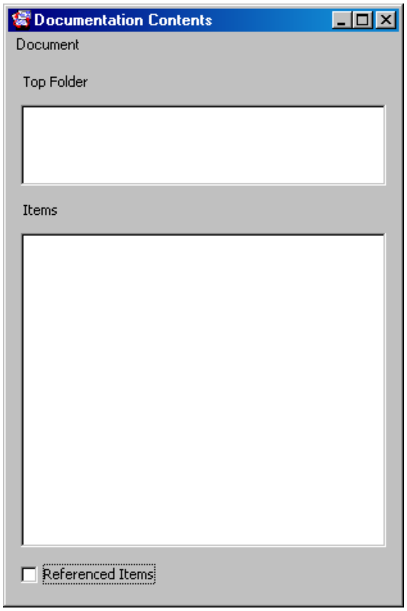
  You can generate documentation in either English or German.

- Select the document format from the "Output Format" combo box.

  The documentation file can be generated in ASCII, RTF, HTML or Postscript format.
- Click **OK**.

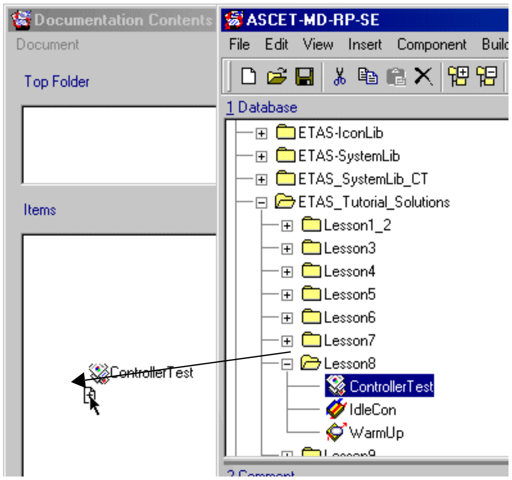
To select the items for generating documentation:

- In the Component Manager, select **Tools** → **Documentation** → **Contents** to open the "Documentation Contents" window.



- Activate the **Referenced Items** option at the bottom of the window to have referenced items included in the documentation contents as you add new items.

- Drag the item or folder from the database browser and drop it onto the "Items" pane of the "Documentation Content" window.



If you drag a folder into the window, all items are shown in the "Items" pane; the name of the top folder is shown in the upper pane.

To change the selection:

- Open the "Documentation Contents" window.
- In the "Items" pane, select a database item.
- Right-click on the item and select **Delete** from the context menu.

  The item is deleted from the items pane, i.e. it will not be included when documentation is generated.
- Select **Move Up** or **Move Down** to change the position of the item in the "Items" pane.

  The position determines the order in which items appear in the generated documentation.

You can store the contents of generated documentation as a list of database items and their order to the file system.

To save/load the documentation contents:

- In the "Documentation Contents" window, select **Document** → **Save Document Items to File**.

  A file selection dialog window opens. The file extension `*.cfg` is

- Select the path and file name for the contents file and click **Save**.

  The file is stored at the specified location.

- Select **Document** → **Load Document Items From File** to load a previously saved documentation content.

To generate documentation:

- In the "Documentation Contents" window, select **Document** → **Generate Document**.

  A naming window opens.

- Enter a file mane and click **OK**.

  The documentation is generated for all items shown in the "Documentation Contents" window.

The generated files are stored in the documentation directory of your ASCET installation. This directory is specified in the "Options" window, "Documentation" node (see chapter 2.2.5 on page 53). Every time you generate documentation any older files with identical names are overwritten.

To view the generated documentation:

- In the "Documentation Contents" window, select **Document** → **Preview Document**.

  The viewer for the type of documentation you have generated opens with the generated file.

7.2 Documentation File Output Formats

All documentation you generate is written to the documentation directory. This directory is specified in the "Options" window, "Documentation" node (see "Options for Automatic Documentation" on page 53). The file names ASCET uses for the documentation files are, by default, always the same, any old files in the directory will be overwritten.

ASCII Format

If you have selected ASCII as the output format, all generated text is written to a file called `text.txt`. If the database item being documented contains any diagrams, they are written to Postscript files, one for each diagram. The text file contains the filename of each diagram. If the class contains C-code, it is incorporated into the text file.

You can load the ASCII text produced by ASCET into any editor or word-processor, and use it as a basis for your own documentation of applications developed with ASCET.

RTF Format

RTF is a standard format for exchanging formatted documents. It can be read by most word processing programs, such as Microsoft Word or WordPerfect. The formatted documentation can be edited or incorporated into other documents. The document is written to a file called `docu.rtf`.

HTML Format

HTML is the standard format of information interchange on the World Wide Web. HTML documents can be viewed with any World Wide Web browser, such as Netscape or Internet Explorer. The HTML source text of the document is written to a file called `docu.htm`. All diagrams are stored in the GIF format, which is also standard on the World Wide Web.

Postscript Format

If you have selected Postscript as the output format, everything will be written to one Postscript file, the diagrams will be positioned inside the text. The output file is called `docmain.ps`.

7.3 Views

In the ASCET block diagram editor and state machine editor, so-called *views* exist. Automatic documentation and diagrams can be designed differently in different views by hiding parts of the information.

Views can be set up individually. The settings include global and element-specific options for presentation in the block diagram or state machine editor and for automatic documentation.

The following information can be shown/hidden globally:

- sequence calls and connectors
- method names for method-local elements, process names for process-local elements

- graphical comments
- A default representation can be defined for various element groups for each view (e.g., all `cont` parameters are displayed as line).

The element-specific settings are described in section "To edit the views of a diagram item:" on page 226.

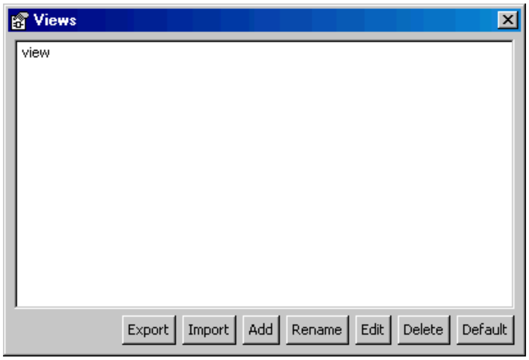
When a view is selected, all global and element-specific settings become active. A diagram's representation in the editor conforms to it's representation in the documentation.

General Remarks

Creating and managing views:

- In the Component Manager, select **Tools** → **Views**.

  The "Views" window opens.



At first, it contains only one view named `view`.

- Click on **Add**.

  A new view is added.

- Enter a name for the view and press <Enter>.
- Click **Rename** to rename a selected view.
- Click **Delete** to remove a selected view.

Note

The buttons **Add**, **Rename**, **Edit**, **Delete** *and* **Default** *are also available as context menu.*

- Click **OK** to close the window and accept the settings.
- Click **Cancel** to close the window without accepting the settings.

To select a default view:

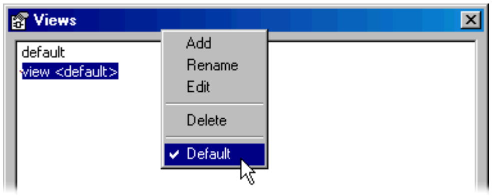
You can choose one of the views as default view. When you open a component in the block diagram editor afterwards, it is shown in the default view.

- In the "Views" window, select one of the available views.
- Click on **Default**

or

- select **Default** from the context menu.

  The selected view is marked as default; the **Default** context menu is tick-marked.



- To undo the selection as default, proceed as follows:
  - Select the default view.
  - Click once more on **Default**

    The default selection is undone and the mark removed. No view is now marked as default.
- To select another view as default, proceed as follows:
  - Select the new view.
  - Click on **Default**

    The marking of the old default view is removed; the new view is selected as default.

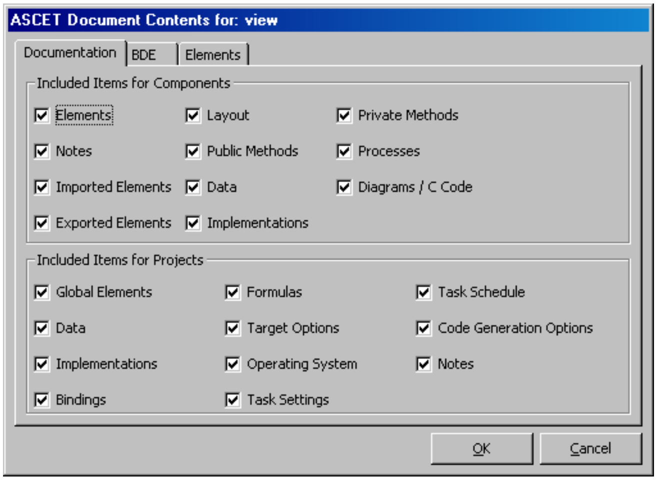
The view selection in the state machine or block diagram editor (see page 184) is not affected by selecting a default.

To edit a view:

You determine, for a given view, which information will be written to the generated documentation.

- In the "Views" window, select the view you want to edit.
- Click on **Edit**.

  The "ASCET Document Contents for " window opens. It contains three tabs, one for the documentation content, one for representation in the block diagram/state machine editor, and one for the representation of special elements.



ASCET Document Contents for: view

Documentation | BDE | Elements

Included Items for Components

- [✓] Elements
- [✓] Layout
- [✓] Private Methods
- [✓] Notes
- [✓] Public Methods
- [✓] Processes
- [✓] Imported Elements
- [✓] Data
- [✓] Diagrams / C Code
- [✓] Exported Elements
- [✓] Implementations

Included Items for Projects

- [✓] Global Elements
- [✓] Formulas
- [✓] Task Schedule
- [✓] Data
- [✓] Target Options
- [✓] Code Generation Options
- [✓] Implementations
- [✓] Operating System
- [✓] Notes
- [✓] Bindings
- [✓] Task Settings

- In these tabs, make your settings for the edited view.

  The meaning of the options is explained in sections "Documentation Options", "Options for Editors", and "Options for Element Types".
- Click on **OK**.

The views you have thus set up are not included in the export of database items. Instead, the "Views" window offers the possibility to export one or more views to an XML file, and to import previously exported views.

To export views:

- In the "Views" window, select one or more views.
- Click on **Export**.

  The "Export File" window opens. It shows all XML files in the ASCET export directory (cf. page 61).
- Enter path and name (with extension `*.xml`) of the export file.
- Click on **Save**

  The selected views are written to the file.

An export file for views has a very simple structure. The name of a view is stored in the `<View>` element, the settings in the tabs of the "ASCET Document Contents" window are stored in the `<Components>` and `<Projects>` elements. Each option in a tab corresponds to an attribute of the respective element; the attribute sequence corresponds to the option sequence first in the left column, then in the right column of the respective tab.

```
<Views>

<View name="view">

<DocumentationSettings>

  <Components notes="true"
       layout="true"
       publicMethods="true"
       privateMethods="true"
       processes="true"
       graphic="true"
       elements="true"
       importedElements="true"
       exportedElements="true"
       data="true"
       implementation="true" />

  <Projects notes="true"
       codegenOptions="true"
       targetOptions="true"
       operatingSystem="true"
```

```
            taskSettings="true"
            taskSchedule="true"
            elements="true"
            bindings="true" data="true"
            implementation="true"
            formulas="true" />
    </DocumentationSettings>
    <BDESettings sequenceCalls="true"
          processNameLocals="false"
          graphicalComments="true" />
    <ElementDefaults>
      <AsLine>
        <Element type="Scalar"
              modelType="Continuous"
              kind="Parameter"
              scope="*"
              existence="Non-Virtual"
              dependency="*"
              memory="*"
              calibration="*"/>
      </AsLine>
      <Invisible></Invisible>
      <Contour></Contour>
      <HideContents></HideContents>
    </ElementDefaults>
    </View>
    </Views>
```

Only the `<View>` element containing the name of the view is mandatory, the other elements and attributes are optional. Missing attributes are set to `true` during import.

When you export several views, separate `<View>` elements are created for each.

To import views:

You can import previously exported views. If an export file contains several views, *all* of them are imported.

- In the "Views" window, click on **Import**.

  A warning opens that existing views (identical names) are overwritten upon import.
- Confirm the warning with **Yes**.

  The "Import File" window opens. It shows all XML files in the ASCET export directory (cf. page 63).
- Select the file you want to import.
- Click on **Open**.

  The views in the selected file are imported in the ASCET database.

Documentation Options

The "Documentation" tab in the "ASCET Document Contents for" window contains the documentation content settings, for components (top) and projects (bottom).

Options for Components:

Elements	All elements of the component are displayed.
Notes	The notes associated with the current view are displayed.
Imported Elements	All imported elements of the component, i.e. messages and global variables, are displayed.
Exported Elements	All exported elements are displayed.
Layout	The block layout of the component is displayed.
Public methods	All methods in a public diagram are displayed.
Data	All data sets of the component are displayed.
Implementations	All implementations of the component are dis-played.
Private Methods	All methods in a private diagram are displayed.
Processes	All processes of the component are displayed (modules only).

Diagrams / C Code	If the component contains block diagrams or state machine diagrams, they are shown. If the component contains C code, that is shown.

Options for Projects:

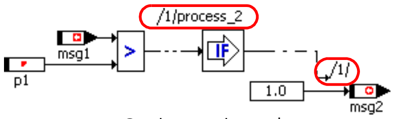
Global Elements	All global elements that are defined within the project are displayed.
Data	All data sets of the project are displayed.
Implementations	All implementations of the project are displayed.
Bindings	Shows to which elements the global elements declared within the project are bound.
Formulas	All global formulae declared in the project are displayed.
Target Options	The current target settings for the project are displayed.
Operating System	all task and process settings made in the operating system editor are displayed.
Task Settings	The settings for each task of the project are displayed.
Task Schedule	Information about the scheduling of the tasks in the project is displayed.
Code Generation Options	The current code generation settings for the project are displayed.
Notes	The notes associated with the current view are displayed.

Options for Editors

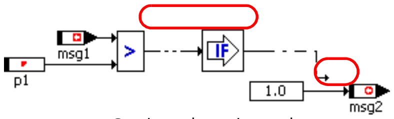
The "BDE" tab contains options for representation in the block diagram and state machine editor. The following options are available:

- **Show Sequence Calls**

  If this option is deactivated for a view, all sequence calls and connectors in the diagrams are hidden in this view.



Option activated

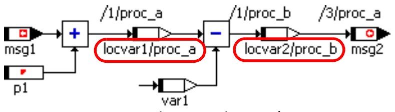


Option deactivated

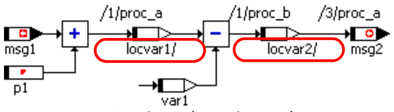
The option is not identical with the **Sequence Calls** → **Show** → ***** and **Sequence Calls** → **Hide** → ***** menu options. The menu options are ineffective if **Show Sequence Calls** is deactivated for the current view.

- **Show Method/Process for Locals**

  If this option is deactivated, the method names in method-local elements and the process names in process-local elements are hidden. The / separator remains visible.



Option activated

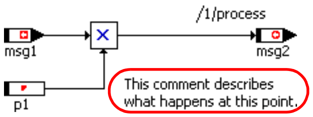


Option deactivated

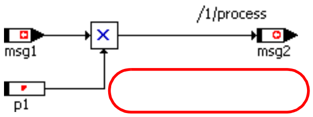
- **Show Graphical Comments**

  If this option is deactivated for a view, all comments in the diagrams are hidden for this view.

  If the option is activated, comments are displayed. If the `Invisible` attribute was selected in the element-specific options for one or more comments, these comments are hidden nonetheless. For more information on element-specific options, see sections "To edit the views of a diagram item:" on page 226 and "Options for Element Types" on page 691.



Option activated



Option deactivated

Options for Element Types

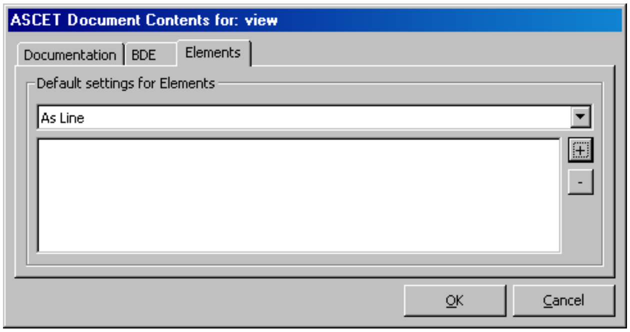
The "Elements" tab is the central place to adjust the representatin of several element groups in the block diagram or state machine editor. These global options apply to all elements of the group, anywhere in the database, for which no individual settings are available.

Note

Diagram elements with an individual setting other than `Normal` *(see "Appearance of Diagram Elements", page 226) are not affected by the global settings.*

To make global settings for element groups:

- Open the "ASCET Document Content for" window, "Elements" tab, for a selected view.



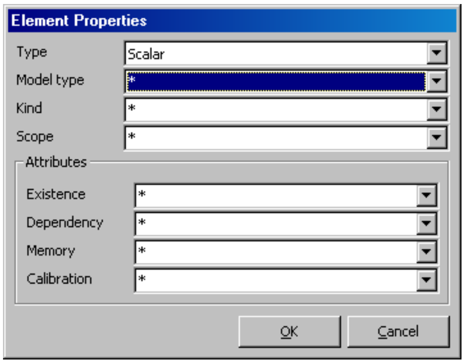
Default settings for Elements: As Line

- In the combo box, select the representation you want to assign to the element group.

  The representations are described on page 226.

- Click on the **+** button to add an element group.

  The "Element Properties" window opens.



Type	Scalar
Model type	*
Kind	*
Scope	*

Attributes

Existence	*
Dependency	*
Memory	*
Calibration	*

- Select type, model type, kind, scope, and the attributes.

  The fields here have the same meaning as the fields with identical names in the element editor (cf. Fig. 4-2 on page 449).

  If no explicit selection is made, (* in a field), the representation selected in the "Elements" tab applies to *all* possible selections.
- Click **OK** to accept the settings and close the window.

  The element group is shown in the "ASCET Document Content" window, "Elements" tab, below the combo box.
- Click **OK** to accept the settings and close the window.

If the element groups for a view are defined in a way that some elements belong to more than one group (e.g., `As Line` for exported system constants, `Hide` for logical system constants), the actual representation results from the processing sequence of the view options in ASCET, it is not clearly defined. In the example, a logical, exported system constant can be displayed as line, or hidden.

Note

When defining element groups, take care to avoid conflicts by multiple assignments of elements to groups.

7.4 Notes

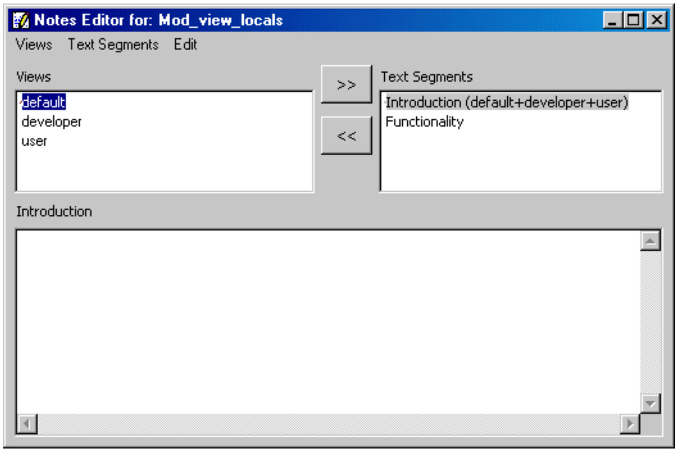
Every folder or database item can have notes associated with it. These are not the same as comments, which are incorporated in block diagrams and printed with those. A note is associated with a folder or database item, and becomes part of the text of the generated documentation. In notes, information can be included that is not part of the functional specification.

To create a note:

- Select the folder or database item that the note is to belong to.

- Select **Component** → **Notes** to add a new note.

  The notes editor opens.



A note consists of a number of text segments, each of which is assigned to a view. A text segment can be assigned to more than one view. That way it is not necessary to re-type the same text for different views. When a document is generated, all text segments that belong to the selected view are included.

To create a text segment:

- In the notes editor, select **Text Segments** → **Add**.
- Enter a name for the text segment and press <ENTER>.

  The name of the text segment is listed in the "Text Segments" pane. The new text segment is assigned to all currently selected views.
- Type in the text for the segment in the text entry pane underneath the "Text Segment" pane.

  The name of the current text segment is shown as the title of the text entry pane.
- Select **Text Segment** → **Rename** to rename the text segment.

- Select **Text Segment** → **Delete** to delete the text segment.

To edit a text segment:

- Select**Edit** → **Cut**, **Copy** or **Paste** to edit the text in the window.

  These operations work with the standard Windows clipboard and can thus be used to exchange text with other applications.
- Select **Edit** → **Select All** to select all the text in the text entry pane.
- Select **Edit** → **Read from File** to read in data from an external file.

  A copy of the source text is created, the source file is not changed by this operation.
- Select **Edit** → **Write to File** to write the text of a note to an external file.

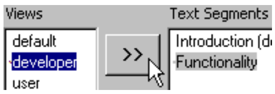
  You are prompted for a path and a filename and the file is stored at that location.
- Select **Edit** → **Save** to save the current text segment.

To assign a text segment to a view:

When it is created, a text segment is assigned to all the views selected in the "Views" pane.

- Select a text segment in the "Text Segment" pane.
- Select a view in the "Views" pane.
- Select **Views** → **Assign Views**

or



- click on the **>>** button.

  A text segment can be assigned to more than one view. To do so, select all views in the "Views" pane while pressing the <CTRL> key.

To deassign a view:

- Select a text segment.
- Select a view in the "Views" pane.

- Select **Text Segment** → **Deassign Views**

or

- click on the **<<** button.

To view text segment assignments:

- Select a view in the "Views" pane.
- Select **Views** → **Select Text Segments**.

  The text segments assigned to the view are highlighted in the "Text Segments" pane.
- Select a text segment in the "Text Segments" pane.
- Select **Text Segments** → **Select Assigned Views**.

  All views the text segment is assigned to are highlighted in the "Views" pane.

Index