DRIVING EMBEDDED EXCELLENCE

ETAS ASCET V6.4

Icon Reference Guide

www.etas.com

Copyright

ASCET V6.4 – Icon Reference Guide R08 EN – 03.2023

Contents

1 Safety and Privacy Information

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

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

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

1.1 Intended Use

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

ASCET provides a multi-paradigm modeling framework, providing integrated support for a number of different modeling notations. These modeling notations abstract from low-level details, separating the concerns of what the system software must do from how it is realized in code executing in the ECU. ASCET can also interface directly with C code as a "low-level" specification language.

ASCET provides a systematic way to augment the high-level specification (referred to as the *physical model*) with the necessary information for target implementation (referred to as the *implementation model*). The implementation model covers the low-level details required to make the model run on target hardware.

The physical and implementation models are clearly separated in ASCET so that the design specification is not corrupted with implementation details that may change from project to project. Maintaining this separation also allows ASCET to support multiple implementation models for a single physical model, keeping the number of model variants low.

1.2 Target Group

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

ASCET users should be familiar with the Microsoft Windows® 10 operating system. They should be able to execute menu commands, enable buttons, etc. Furthermore, ASCET users should be familiar with the Windows file storage system, especially the connections between files and directories. They have to know how to use the basic functions of the Windows File Manager and Program Manager or the Windows Explorer, respectively, and they should be familiar with the "drag-and-drop" functionality.

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

1.3 Classification of Safety Messages

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

⚠ **DANGER**

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

⚠ **WARNING**

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

⚠ **CAUTION**

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

NOTICE

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

1.4 Safety Information

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

WARNING

Harm or property damage due to unpredictable behavior of vehicle or test bench

Wrongly initialized NVRAM variables (NV variables) can lead to unpredictable behavior of a vehicle or a test bench. This behavior can cause harm or property damage.

ASCET projects that use the NVRAM possibilities of ASCET-RP targets expect a *user-defined* initialization that checks whether all NV variables are valid for the current project, both individually and in combination with other NV variables. If this is not the case, all NV variables have to be initialized with their (reasonable) default values.

Due to the NVRAM saving concept, this is *absolutely necessary* when projects are used in environments where any harm to people and equipment can happen when unsuitable initialization values are used (e.g. in-vehicle-use or at test benches).

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

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

- In electronic form on the DVD: `Documentation\ETAS Safety Advice.pdf`
- The ASCET V6.4 safety manual, available at ETAS upon request

1.5 Privacy Notice

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

1.5.1 Data Processing

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

1.5.2 Data and Data Categories

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

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

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

- The workspace of the product points to a directory without personal reference.
- All settings in the product (see the menu function **Tools > Options** in the product) refer to directories and file names without personal reference.
- All project settings in the projects (see the menu function **File > Properties** in the ASCET project editor) refer to directories and file names without personal reference.
- Windows environment variables (such as the temporary directory) refer to directories without personal reference because these environment variables are used by the product.

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

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

- User data: UserID
- Communication data: IP address

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

- Problem Report, see section 1.5.4 below

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

- User data: UserID

1.5.3 Technical and Organizational Measures

This product does not itself encrypt the personal data that it records. Please ensure that the data recorded is secured by means of suitable technical or organizational measures in your IT system, e.g. by using classic anti-theft and access protection.

Personal data in generated files can be deleted by tools in the operating system.

1.5.4 Description of Problem Report

Purpose:

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

Personal Data:

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

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

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

This automatically created zip file contains the following:

- product-related log files created at installation time (necessary for uninstall action)
- ETAS log files stored in the ETAS log files directory matching the file name pattern `*.log`
- recursive registry export of ETAS (32bit)-key (and sub keys): `HKEY_CURRENT_USER\Software\ETAS`
- registry export of ETAS (32bit)-key (and sub keys): `HKEY_LOCAL_MACHINE\Software\ETAS`

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

2 About ASCET

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

This manual lists the icons and symbols that appear in classes and modules.

2.1 Finding Out More

The ASCET Icon Reference Guide contains the following chapters:

- **"About this Document"**

  This chapter explains how information is presented in this user guide.
- **"About ASCET"** (this chapter)

  This chapter contains general information such as documentation conventions.
- **"Operators in Block Diagrams"**

  This chapter lists all block diagram operators and their icons.
- **"Control Flow Elements in Block Diagrams"**

  This chapter lists all control flow elements and their icons.
- **"Elements in Block Diagrams"**

  This chapter lists all block diagram elements and their icons.
- **"Miscellaneous Icons"**

  This chapter lists the remaining icons and their meaning.
- **"Contact Information"**

  Contact information of the ETAS subsidiaries.

The following further documentation is available when you installed the respective ASCET product:

- ASCET base system
  - ASCET Getting Started (this manual; `ASCET V6.4 Getting Started.pdf`)
  - ASCET Installation Guide (`ASCET V6.4 Installation.pdf`)
  - ASCET online help; accessible via the **Help** menu and <F1> in the various ASCET windows
  - ASCET AUTOSAR User Guide (`ASCET V6.4 AUTOSAR User Guide.pdf`)
  - AUTOSAR to ASCET Importer User Guide (`ASCET V6.4 AUTOSAR To ASCET Converter User Guide.pdf`)

NOTE

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

- ASCET-RP
  - user guide (`ASCET-RP V6.4 User Guide.pdf`)
  - separate online help; accessible via the **Help** menu and <F1> in the Hardware Configurator
- ASCET-SE
  - ASCET-SE user guide (`ASCET-SE V6.4 User Guide.pdf`)
  - EHOOKS Add-On user guide (`ASCET-SE V6.4 EHOOKS Add On User Guide.pdf`)
- ASCET-SCM
  - online help (integrated in the main ASCET online help)
- ASCET-DIFF
  - ASCET-DIFF installation guide
  - separate online help; accessible via the **Help** menu and <F1> in the ASCET-DIFF windows

3 Operators in Block Diagrams

This chapter lists the operators available in ASCET block diagrams.

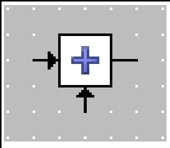
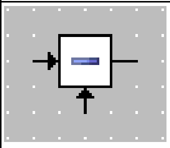
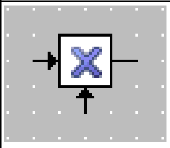
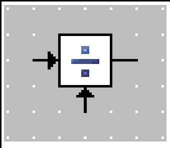
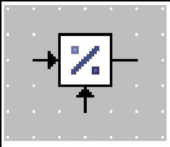
- arithmetic operators (section 3.1 on page 8)
- comparison operators (section 3.2 on page 9)
- logical operators (section 3.3 on page 10)
- multiplex operator (section 3.4 on page 10)
- case operator (section 3.5 on page 11)
- miscellaneous operators (section 3.6 on page 11)

3.1 Arithmetic Operators

The meaning of the operators is the same as in ESDL. The following operators are available:

- Addition, Subtraction, Multiplication, Division, Modulo

The addition and multiplication operators can have 2 to 20 arguments. The subtraction, division and modulo operators have only two arguments.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Addition	+	+		Returns the sum of the inputs.
Subtraction	−	−		Returns the difference between the upper and the lower input.
Multiplication	×	×		Returns the product of the inputs.
Division	÷	÷		Returns the quotient of the upper and the lower input.
Modulo	%	%		Returns the remainder of the division upper / lower input.

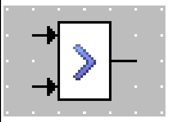
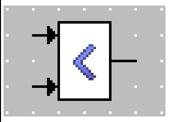
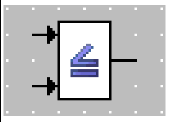
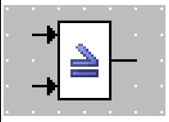
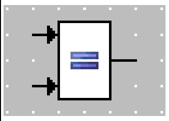
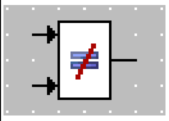
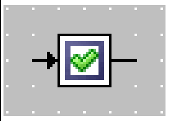
a. By default, no arithmetic operators are shown in the "Navigation" tab.

3.2 Comparison Operators

The comparison operators are identical to their counterparts in ESDL. The following comparison operators are available:

- Greater
- Smaller
- Smaller or Equal
- Greater or Equal
- Equal
- Not Equal

Each operator has 2 arguments. The Equal and Not Equal operators can be applied to arithmetic and non-arithmetic elements.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Greater	>	>		Returns `true` if the upper input is greater than the lower input. Returns `false` otherwise.
Smaller	<	<		Returns `true` if the upper input is smaller than the lower input. Returns `false` otherwise.
Smaller or Equal	≤	≤		Returns `true` if the upper input is smaller than or equal to the lower input. Returns `false` otherwise.
Greater or Equal	≥	≥		Returns `true` if the upper input is greater than or equal to the lower input. Returns `false` otherwise.
Equal	=	=		Returns `true` if the upper input is equal to the lower input. Returns `false` otherwise.
Not Equal	≠	≠		Returns `true` if the upper input is not equal to the lower input. Returns `false` otherwise.
Verify	✓	✓		Returns `true` if original and complement of a redundant element are consistent. Returns `false` otherwise.

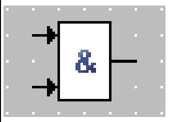
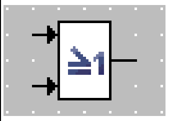
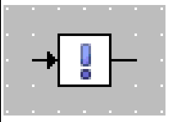
a. By default, no comparison operators are shown in the "Navigation" tab.

3.3 Logical Operators

The logical operators are identical to their counterparts in ESDL. The following logical operators are available:

- And
- Or
- Not

The And and Or operators can have 2 to 20 arguments, the Not operator has one argument.

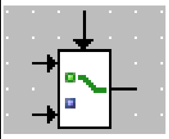
name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
And	&	&		Returns `true` if all inputs are `true`. Returns `false` if at least one input is `false`.
Or	≥1	≥1		Returns `true` if at least one input is `true`. Returns `false` if all inputs are false.
Not	!	!		Returns `true` (`false`) if the input is `false` (`true`).

a. By default, no logical operators are shown in the "Navigation" tab.

3.4 Multiplex Operator

The conditional operator `( ? : )` is named *Multiplex* operator (for short: *Mux*) in the graphical representation.

The multiplex operator can be used with 2 to 20 arguments, however, the identical functionality of a multi-argument Mux operator can be built as a cascade of several 2-argument Mux operators.

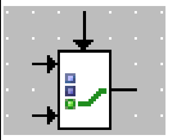
name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Mux				The input on the top is the condition, the inputs on the left are the values. The Mux operator returns the upper (lower) left input if the condition is `false` (`true`).

a. By default, no Mux operators are shown in the "Navigation" tab.

3.5 Case Operator

The *Case* operator is a special case of the conditional operator. It does not take a logical value, but a switch value of discrete type. The Case operator has `n` arguments (`n = 2..20`), `n-1` of which are numbered consecutively. The last argument is the default case.

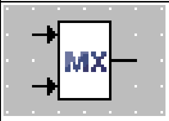
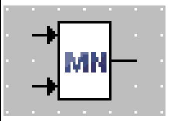
Depending on the switch value, one of the arguments is selected. If the switch value is 1, the first argument is returned, if it is 2 the second is returned, and so on. If the switch value is < 1, or `n`, or > `n`, the last argument is returned.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Case				The input on the top is the switch value, the inputs on the left are the arguments.

a. By default, no Case operators are shown in the "Navigation" tab.

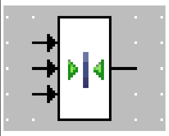
3.6 Miscellaneous Operators

Max and Min Operators: The Max and Min operators can have 2 to 20 arguments; they can be applied only to arithmetic elements.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Max	MX	MX		Returns the largest input value.
Min	MN	MN		Returns the smallest input value.

a. By default, no Min/Max operators are shown in the "Navigation" tab.

Between Operator: The *Between* operator checks if the argument value (upper input) lies between the limiters min (middle input) and max (lower input). If this is the case, the logical return value is true, otherwise it is set to false.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Between				Returns `true` (`false`) if the argument lies (does not lie) between min and max.

a. By default, no Between operators are shown in the "Navigation" tab.

Absolute Operator: Argument and return value of the *Absolute* operator have to be both either cont or discrete.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Absolute				Returns the absolute value of the argument.

a. By default, no Absolute operators are shown in the "Navigation" tab.

Negation Operator: Argument and return value of the *Negation* operator can be cont or discrete; if the argument is cont, the type of the return value should be the same.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Negation				Returns the negative value of the argument.

a. By default, no Negation operators are shown in the "Navigation" tab.

Conversion Operator: The *Conversion* operator allows to convert scalar elements of numerical or enumeration type to limitInt or wrapInt types. The function of the convert operator is determined either via the button used to create the operator or via the Conversion Type submenu in the operator's context menu.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Conversion Limit	[ ]	[ ]	[ ]	Converts the argument to a limitInt type.
Conversion Wrap-Around				Converts the argument to a wrapInt type.

a. By default, no Conversion operators are shown in the "Navigation" tab.

Size Operators: The *Size X* and *Size Y* operators allow to access the X and Y (if applicable) size of arrays, matrices, characteristic curves/maps and distributions.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Size X				Returns the X size of an array, matrix, characteristic line/map or distribution.
Size Y				Returns the Y size of a matrix or characteristic map.

a. By default, no Size operators are shown in the "Navigation" tab.

Assert Operator: The *Assert* operator allows to specify lower and upper bound of an interval. This interval is then used by the code generator as the result interval of the Assert operator; the calculated interval of the operand is overwritten.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Assert				Allows to specify lower and upper bound of an interval.

a. By default, no Assert operators are shown in the "Navigation" tab.

Communication Status Operator: The *Communication Status* operator is used to perform a request for the communication status of implicit communication in an AUTOSAR environment. You can use the status to determine if an implicit read access was successful.

In a non-AUTOSAR environment, the status operator is always generated to `OK`. Code that becomes obsolete in that case is removed during code optimization. The same happens in an AUTOSAR environment if the RTE does not support the configured error code.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Communication status				Returns the communication status for implicit communication in an AUTOSAR environment.

a. By default, no Communication Status operators are shown in the "Navigation" tab.

4 Control Flow Elements in Block Diagrams

This chapter lists the control flow elements available in ASCET block diagrams.

The following control flow statements are available in block diagrams:

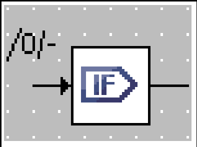
- If...Then (section 4.1 on page 14)
- If...Then...Else (section 4.2 on page 15)
- Switch (section 4.3 on page 15)
- While (section 4.4 on page 16)
- Break Statement (section 4.5 on page 16)

All control flow statements except Break evaluate a logical expression and, depending on the result, activate a control flow branch which may contain several statements. The statements represented by sequence calls are connected to the control flow by connectors.

The Break statement can be used to exit immediately from each of the other control flow elements and return to another enclosing statement or to the remainder of the model.

4.1 If...Then

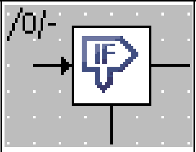
The *If...Then* statement evaluates a logical expression. The control flow output is connected to one or more sequence calls which are triggered whenever the control flow branch is activated. Whenever the input expression evaluates to `true`, the connected sequence calls are executed.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
If...Then		/		Activates a control flow branch (connected to the output) if the input is `true`.

a. By default, no If...Then blocks are shown below the "Graphic Blocks" node in the "Navigation" tab.

4.2 If...Then...Else

If...Then...Else is similar to If...Then, but has two control flow branches. Depending on the value of the logical expression, one of the branches is executed.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
If...Then... Else		/ (if) / (else)		Activates the control flow branch on the right (connected to the output) if the input is `true`. Activates the control flow branch at the bottom if the input is `false`.

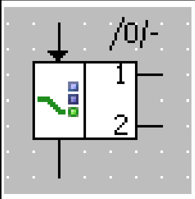
a. By default, no If...Then...Else blocks are shown below the "Graphic Blocks" node in the "Navigation" tab.

4.3 Switch

The *Switch* construct is similar to the Case operator (cf. section 3.5 on page 11). A Switch evaluates a signed discrete or unsigned discrete value and, depending on that value, activates different control flow branches. These branches are separated from each other, so that a "fall through" like in the switch construct in C is not possible.

A switch can have 2 to 20 branches.

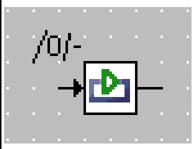
For each alternative, the value for the branch can be defined by the user. The last branch at the bottom is the default branch.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Switch		/ (case) / (default)		Activates the control flow branch whose value is the same as the input value. Activates the default branch (at the bottom) if no branch value equals the input.

a. By default, no Switch blocks are shown below the "Graphic Blocks" node in the "Navigation" tab.

4.4 While

The *While* loop is the only loop construct available in block diagrams . Care has to be taken to avoid infinite loops or loops unsuitable for real-time applications.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
While				Activates the control flow branch if the input value is `true`. The operation is executed as long as the input value remains `true`; the input value should be manipulated in the while loop.

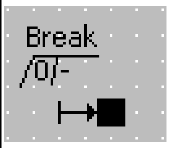
a. By default, no While blocks are shown below the "Graphic Blocks" node in the "Navigation" tab.

4.5 Break

The *break* operator in the block diagram editor behaves similar to a C language return statement.

NOTE

The **break** operator in the block diagram editor behaves differently from the break statement in ESDL.

name	icon in			remarks
	palette/ toolbar	"Navigation" tab[a]	block diagram	
Break				In *method*: Causes an immediate return from the method. The user is responsible for the correct setting of any return values before break is executed. In *process*: Causes a deferred exit (i.e. all send messages are sent before the exit occurs).

a. By default, no Break blocks are shown below the "Graphic Blocks" node in the "Navigation" tab.

5 Elements in Block Diagrams

This chapter lists the elements available in ASCET block diagrams.

- section 5.1 "Scalar Elements" on page 17
- section 5.2 "Composite Elements" on page 20
- section 5.3 "Complex Elements (Included Components)" on page 25
- section 5.4 "Real-Time Elements" on page 25
- section 5.5 "Signature Elements" on page 28
- section 5.6 "Miscellaneous Elements" on page 31

5.1 Scalar Elements

Several scalar elements are available in ASCET block diagrams:

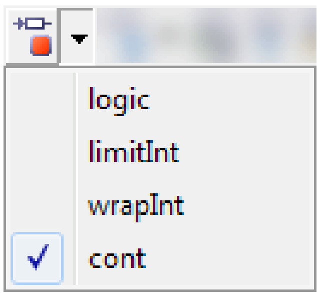
- variables (section 5.1.1 on page 17)
- parameters (section 5.1.2 on page 18)
- literals (section 5.1.3 on page 19)
- constants and system constants (section 5.1.4 on page 20)

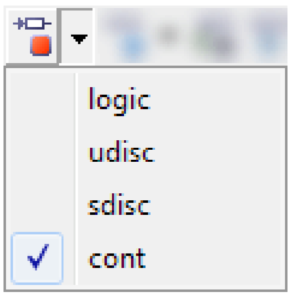
5.1.1 Variables

The following types of scalar variables are available in ASCET block diagrams:

- Logic
- Limited integer
- Wrap-around integer
- Signed discrete
- Unsigned discrete
- Continuous
- Enumeration

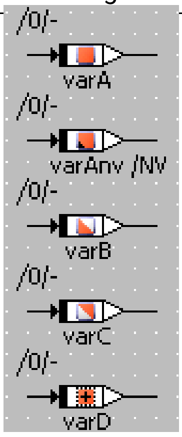
The "Elements" palette provides a button for each variable, the "Elements" toolbar provides a single button with a sub-menu for the type of the numeric variable. The content of the submenu depends on the **Use signed/unsigned discrete types** option in the "Appearance\Editors" node of the ASCET options window.





In the block diagram, the type of a variable is not visible, only kind (variable, virtual variable) and scope (exported/imported/local) and some properties (virtual/nonvirtual, volatile/nonvolatile); see chapter 6 on page 34 for details.

The type of a variable is shown in the "Tree Pane".

name	palette icon	block diagram icon[a]	"Tree Pane" icon
Logic Variable	L🟥		L 🟥 l_var::log
Limited Integer Variable (Signed Discrete Variable)	I🟥 (S🟥)		I 🟥 i_var::limitInt S 🟥 s_var::sdisc
Wrap-Around Integer Variable (Unsigned Discrete Variable)	W🟥 (U🟥)		W 🟥 w_var::wrapInt U 🟥 u_var::udisc
Continuous Variable	C🟥		C 🟥 c_var::cont
Enumeration Vari-able	E🟥		E 🟥 e_var::enum_s

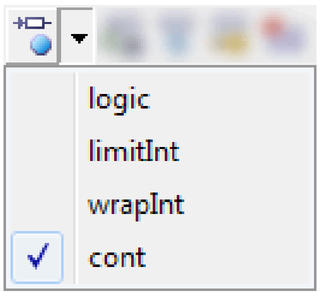
a. The patterns of the red squares indicate the scope (cf. section 6.1) and some properties (cf. section 6.2) of the variables.

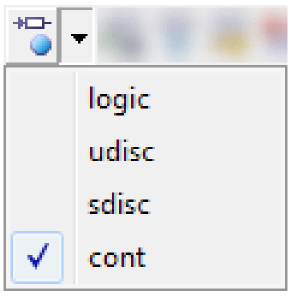
5.1.2 Parameters

The following types of scalar parameters are available in ASCET block diagrams:

- Logic,
- Limited integer,
- Wrap-around integer,
- Signed discrete,
- Unsigned discrete,
- Continuous,
- Enumeration

The "Elements" palette provides a button for each parameter, the "Elements" toolbar provides a single button with a sub-menu for the type of the numeric parameter.





In the block diagram, the type of a parameter is not visible, only kind and scope (exported/imported/local) and some properties (e.g., normal/dependent); see chapter 6 on page 34 for details.

The type of a parameter is shown in the "Tree Pane".

name	palette icon	block diagram icon[a]	"Tree Pane" icon
Logic Parameter	L	parAnv	L l_par::log
Limited Integer Parameter (Signed Discrete Parameter)	I (S)	parBnv	I i_par::limitInt S s_par::sdisc
Wrap-Around Integer Parameter (Unsigned Discrete Parameter)	W (U)	parCnv parDnv	W w_par::wrapInt U u_par::udisc
Continuous Parameter	C	parEnv	C c_par::cont
Enumeration Parameter	E		E e_par::enum_s

a. The patterns of the blue circles indicate the scope (cf. section 6.1) and some properties (cf. section 6.2) of the parameter.

5.1.3 Literals

Literals are strings that represent a fixed value of a basic scalar type which can be used in any expression. The value of a literal is either a number (discrete or continuous), a character string, or one of the values `true` or `false` (logical).

The following literals are predefined in ASCET block diagrams:

- Enumeration literal
- Logic literal `true`
- Logic literal `false`
- Continuous literal `0.0`
- Continuous literal `1.0`

name	palette / toolbar icon	block diagram icon	"Tree Pane" icon[a]
Enumeration Literal	ENUM	string	ENUM string
Logic Literal true	TRUE	true	ENUM true
Logic Literal false	FALSE	false	ENUM false
Continuous Literal 0.0	0.0	0.0	ENUM 0.0
Continuous Literal 1.0	1.0	1.0	ENUM 1.0

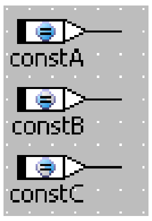
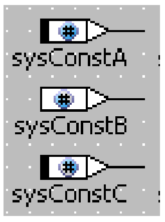
a. Literals are shown only in the "Navigation" tab. They do not appear in the "Outline" tab.

5.1.4 Constants and System Constants

Constants and system constants cannot be created via palette or toolbar buttons. They are created with the parameter or variable buttons (cf. section 5.1.1 and section 5.1.2) and the appropriate selection (i.e. `Constant` or `System Constant`) in the "Kind" combo box of the properties editor.

Constants and system constants can be of scope exported, imported or local; they can be of the types logic, limited/wrap-around integer, signed/unsigned discrete, continuous, enumeration (see also chapter 6 on page 34).

The representation of constants and system constants in the "Tree Pane" is similar to that of parameters (section 5.1.2), except for the overlay icon (**=** or **#**). User-defined system constants can have user-defined overlay icons; see the online help for details.

name	palette / toolbar icon	block diagram icon[a]	"Tree Pane" icon	Remarks
Constant	n/a		C = cnst::cont	Constants are created as a define statement in the generated C code. However, they are not necessarily explicitly visible in the generated code.
System Constant	n/a		C # sCnst::cont	System constants are used like constants, and also created as define statements. System constants can be implemented. They are always explicitly visible in the generated code.

a. The patterns of the blue circles indicate the scope (cf. section 6.1) and some properties (cf. section 6.2) of the constants and system constants.

5.2 Composite Elements

Several composite, i.e. non-scalar elements are available in ASCET block diagrams:

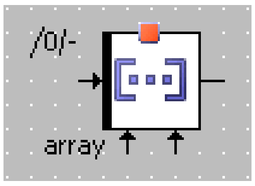
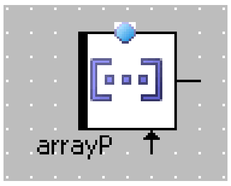
- arrays (section 5.2.1 on page 21)
- matrices (section 5.2.2 on page 21)
- characteristic lines and maps (section 5.2.3 on page 22)

Composite elements can be variables or parameters of any scope (cf. section 6.1). Arrays and matrices can also be used as messages; these are described in section 5.4.1 on page 25.

5.2.1 Arrays

An *array* is a one-dimensional, indexed set of variables or parameters which have the same scalar type (logic, limited/wrap-around integer, signed/unsigned discrete, continuous). This type is not visible in the diagram. In the "Tree Pane", no special icon indicates the array type, it is visible only textually in the "Outline" tab.

The position of a scalar value within an array is indicated by its associated index value which must be a non-negative integer.

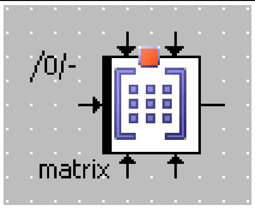
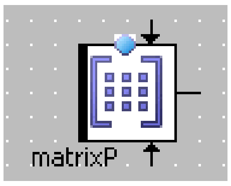
name	palette / toolbar icon	block diagram icon	"Tree Pane" icon
Array (variable)			🟥 array::array[cont] 🟥 arrayEnum::array[Enumer] 🟥 arrayRecord::array[Rec]
Array (parameter)			🔵 arrayP::array[cont] 🔵 arrayPEnum::array[Enumer] 🔵 arrayPRecord::array[Rec]

For the icon differences induced by scope, see section 6.1 on page 34.

5.2.2 Matrices

A *matrix* is a two-dimensional, indexed set of variables or parameters which have the same scalar type (logic, limited/wrap-around integer, signed/unsigned discrete, continuous). This type is not visible in the diagram. In the "Tree Pane", no special icon indicates the matrix type, it is visible only textually in the "Outline" tab.

The position of a scalar value within a matrix is indicated by its associated X and Y index values, which must be non-negative integers.

name	palette / toolbar icon	block diagram icon	"Tree Pane" icon
Matrix (variable)			🟥 matrix::mat[cont] 🟥 matrixEnum::mat[Enumer] 🟥 matrixRecord::mat[Rec]
Matrix (parameter)			🔵 matrixP::mat[cont] 🔵 matrixPEnum::mat[Enumer] 🔵 matrixPRecord::mat[Rec]

For the icon differences induced by scope, see section 6.1 on page 34.

5.2.3 Characteristic Lines and Maps

To support nonlinear control engineering, characteristic lines and maps are available in ASCET block diagrams. They are used to describe a value in dependence of one or two other values.

Characteristic lines and maps are available in the varieties *normal*, *fixed*, and *group*.

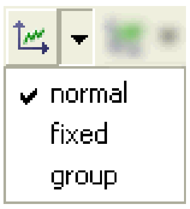
A *fixed* characteristic line/map has an equidistant distribution, i.e. the sample points have a constant distance from each other.

A *group* characteristic line/map does not contain a sample point distribution, but references an external distribution (cf. page 23) of sample points.

Characteristic Lines

A *characteristic line* is represented as a one-dimensional table of sample points, each of which is associated with a sample value. The sample points represent the X axis of a function graph, the sample values represent the curve being described.

The "Elements" palette provides a combo box and a button for characteristic lines, the "Elements" toolbar provides a single button with a submenu for the variety.



In the block diagram, the types of sample points and values of a characteristic line are not visible, only kind, scope (exported/imported/local) and some properties; see chapter 6 "Miscellaneous Icons" for details. In the "Tree Pane", no special icons indicate the types, they are visible only textually in the "Outline" tab.

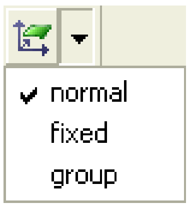
Green (red) arrows indicate that the sample point distribution is strictly monotonic increasing (decreasing). No arrow is shown if strict monotony is not given.

name	palette icon	block diagram icon	"Tree Pane" icon
OneD Table (normal); strictly monotonic increasing	Normal +	CharLine	🔵 CharLine::1D[cont->cont] 🟥 CharLineV::1D[cont->cont]
OneD Table (normal); strictly monotonic decreasing		charLineDown	
OneD Table (normal); no strict monot-ony		charLineMisc	
OneD Table (fixed)	Fixed +	fixedCharLine	
OneD Table (group)	Group +	groupLine (X_dist)	
Distribution (required for group char. line)		/0/- X_dis	🔵 X_dis::distrib[cont]

For the icon differences induced by scope, see section 6.1 on page 34.

Characteristic Maps

A *characteristic map* is represented as a two-dimensional table of sample points; each pair of sample points is associated with a sample value.

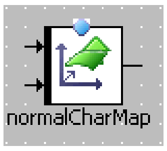
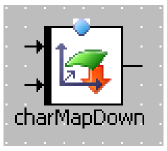
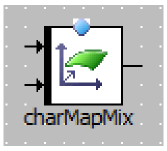
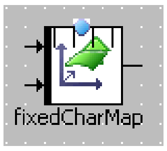
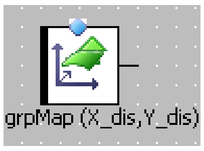
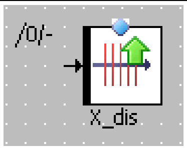


The "Elements" palette provides a combo box and a button for characteristic maps, the "Elements" toolbar provides a single button with a submenu.

In the block diagram, the types of sample points and values of a characteristic map is not visible, only kind, scope (exported/imported/local) and some properties; see chapter 6 on page 34 for details. In the "Tree Pane", no special icons indicate the types, they are visible only textually in the "Outline" tab.

Green (red) arrows indicate that the X sample point distribution is strictly monotonic increasing (decreasing). No arrow is shown if strict monotony is not given.

The monotony behavior of the Y sample point distribution is not shown in the block diagram.

name	palette icon	block diagram icon	"Tree Pane" icon
TwoD Table (normal); X axis strictly monotonic increasing	Normal +		CharMap::2D[cont,cont->cont] charMapV::2D[cont,cont->cont]
TwoD Table (normal); X axis strictly monotonic decreasing			
TwoD Table (normal); no strict monot-ony for X axis			
TwoD Table (fixed)	Fixed +		
TwoD Table (group)	Group +		
Distribution (2 required for group char. map)			X_dis::distrib[cont]

For the icon differences induced by scope, see section 6.1 on page 34.

5.3 Complex Elements (Included Components)

In a block diagram, an included component, or complex element, is represented by the component's layout.

In the "Tree Pane", an included component is represented by its component type icon.

component type	icon	remarks
module[a]	🟥 Module_BDE::Module_BDE 🟥 Module_C::Module_C 🟥 Module_ESDL::Module_ESDL	Modules can only be included in other modules.
class[a]	🟥 Class_BDE::Class_BDE 🟥 Class_C::Class_C 🟥 Class_ESDL::Class_ESDL	
state machine	🟥 State_Machine::State_Machine	State machines, Boolean tables and conditional tables are special classes.
Boolean table	🟥 Class_BoolTab::Class_BoolTab	
conditional table	🟥 Class_CondTab::Class_CondTab	
record	🟥 Record::Record	

a. The small blue boxes with letters denote the item type, i.e. **B**lock diagram, **C** code, or **E**SDL.

5.4 Real-Time Elements

5.4.1 Messages

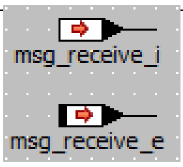
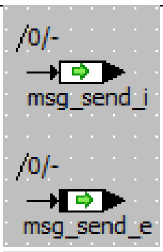
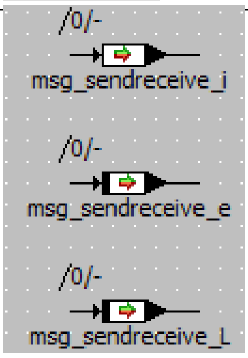
Messages of scalar, composite (array, matrix) or complex (record) type form the input and output variables of processes and are used for inter-process communication. Three types of messages are available in ASCET block diagrams:

- Receive messages
- Send messages
- Send & Receive messages

Scalar messages can be of the same type as variables and parameters, i,e, logic, limited/wrap-around integer, signed/unsigned discrete, continuous, enumeration. As for variables and parameters, the message type is not visible in the diagram. In the "Tree Pane", no special icon indicates the message type, it is visible only textually in the "Outline" tab.

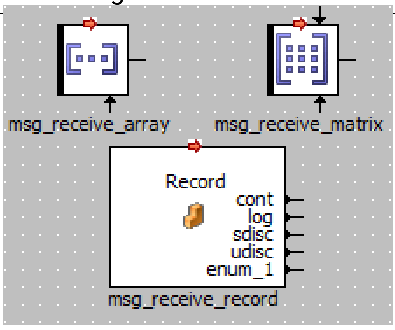
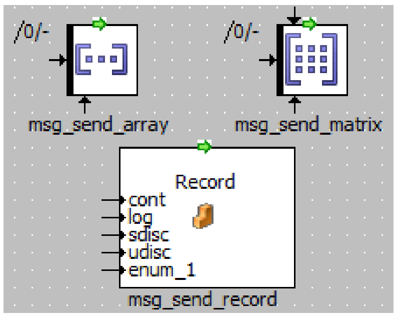
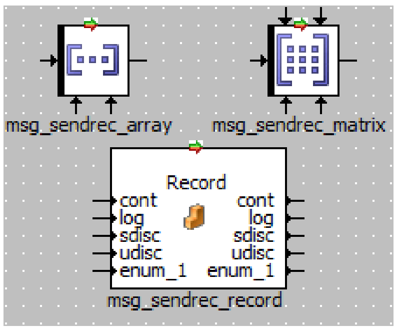
All messages can be of scope exported or imported, only send & receive messages can be of scope local (cf. section 6.1 on page 34). However, the scope is not shown in the block diagram or in the "Tree Pane", it is visible only in the Properties editor.

Scalar messages:

name	palette / toolbar icon	block diagram icon	"Tree Pane" icon[a]
Receive Message			msg_receive_e::cont msg_receive_i::cont
Send Message			msg_send_e::cont msg_send_i::cont
Send & Receive Message		 /0/- msg_sendreceive_i /0/- msg_sendreceive_e /0/- msg_sendreceive_L	msg_sendrec_e::cont msg_sendrec_i::cont msg_sendrec_L::cont

a. The letter icons (/ /) are shown only in the "Outline" tab.

Non-scalar messages: Non-scalar messages cannot be created via the "Elements" toolbar or palette. See the online help for further information.

name	block diagram icon	"Tree Pane" icon[a]
Receive Message of array / matrix / record type	 msg_receive_array msg_receive_matrix Record cont log sdisc udisc enum_1 msg_receive_record	msg_receive_array::array[cont] msg_receive_matrix::mat[cont] msg_receive_record::Record cont::cont enum_1::Enumeration log::log sdisc::sdisc udisc::udisc
Send Message[b] of array / matrix / record type	 /0/- msg_send_array /0/- msg_send_matrix Record cont log sdisc udisc enum_1 msg_send_record	msg_send_array::array[cont] msg_send_matrix::mat[cont] msg_send_record::Record cont::cont enum_1::Enumeration log::log sdisc::sdisc udisc::udisc
Send & Receive Message[a] of array / matrix / record type	 msg_sendrec_array msg_sendrec_matrix Record cont log sdisc udisc enum_1 msg_sendrec_record	msg_sendrec_array::array[cont] msg_sendrec_matrix::mat[cont] msg_sendrec_record::Record cont::cont enum_1::Enumeration log::log sdisc::sdisc udisc::udisc

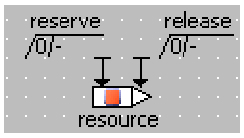
a. The letter icons (/ /) are shown only in the "Outline" tab.

b. Each input pin of the record message has a sequence call. These are hidden in the images for clarity.

5.4.2 Resources

A resource represents a part of an application that can only be used exclusively. In order to access a resource, there are two methods:

- `void reserve()`: the resource is reserved, i.e. access to it is blocked.
- `void release()`: the resource is released, i.e. access to it is granted again.

name	palette / toolbar icon	block diagram icon	"Tree Pane" icon
Resource			resource::resource

5.4.3 dT Parameter

In control engineering applications the result of the calculations within a component often depends on the value of the sampling rate. ASCET provides the system parameter *dT* for uniformly describing the algorithms for all sampling rates. The value of this parameter is provided by the operating system and represents the time difference since the last activation of the currently active task.

In a block diagram component, dT is of scope imported. In a project, dT is of scope exported.

name	palette / toolbar icon	block diagram icon	"Tree Pane" icon
dT		dT	dT::dT (in block diagram) dT::dT (in project)

5.5 Signature Elements

ASCET block diagrams can contain elements of method and process signatures.

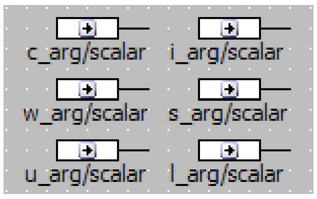
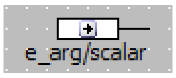
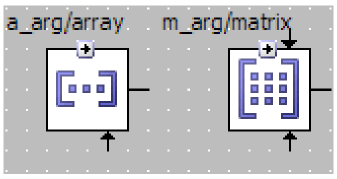
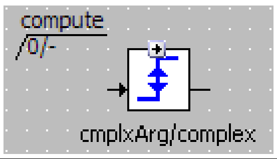
- Method signature elements (section 5.5.1 on page 28)
- process signature elements (section 5.5.2 on page 31)

5.5.1 Method Signature Elements

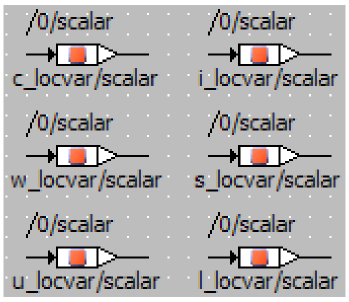
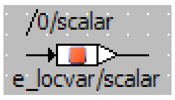
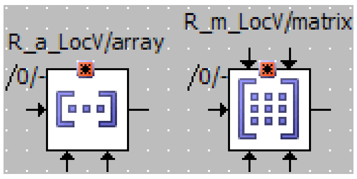
The signature elements of a method can only be created in the signature editor. A method can have the following signature elements:

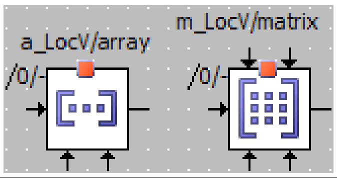
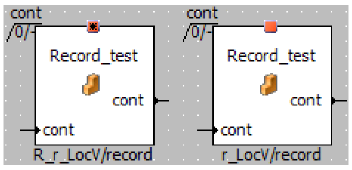
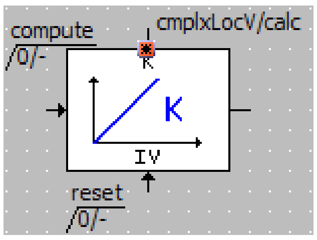
- arguments ("Method Arguments" on page 29)
- local variables ("Local Variables (Method)" on page 29)
- return value ("Return Value" on page 30)

5.5.1.1 Method Arguments

Argument type	block diagram icon	Icon in "Tree Pane" and signature editor
scalar (logic, limited/ wrap-around integer, signed/ unsigned dis-crete, continu-ous)	 c_arg/scalar i_arg/scalar w_arg/scalar s_arg/scalar u_arg/scalar l_arg/scalar	c_arg::cont [In] i_arg::limitInt [In] l_arg::log [In] s_arg::sdisc [In] u_arg::udisc [In] w_arg::wrapInt [In]
enumeration		e_arg::Enumeration [In]
composite		a_arg::array[cont] [In] m_arg::mat[cont] [In]
complex		cmplxArg::EdgeBi [InOut]

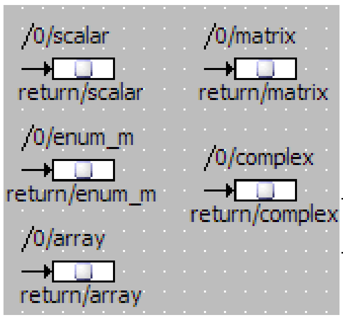
5.5.1.2 Local Variables (Method)

variable type	block diagram icon	icon in "Tree Pane" and signature editor
scalar (logic, limited/ wrap-around integer, signed/ unsigned dis-crete, continu-ous)	 /0/scalar c_locvar/scalar /0/scalar i_locvar/scalar /0/scalar w_locvar/scalar /0/scalar s_locvar/scalar /0/scalar u_locvar/scalar /0/scalar l_locvar/scalar	c_locvar::cont i_locvar::limitInt l_locvar::log s_locvar::sdisc u_locvar::udisc w_locvar::wrapInt
enumeration		e_locvar::Enumeration
composite (explicit reference)		R_a_LocV::array[cont] R_m_LocV::mat[log]

variable type	block diagram icon	icon in "Tree Pane" and signature editor
composite (instance)		a_LocV::array[cont] m_LocV::mat[cont]
record (explicit reference / instance)	 cont /0/- Record_test cont cont R_r_LocV/record cont /0/- Record_test cont cont r_LocV/record	r_LocV::Record_test R_r_LocV::Record_test
complex (except record; always explicit reference)	 compute /0/- cmplxLocV/calc K K IV reset /0/-	cmplxLocV::IntegratorK

5.5.1.3 Return Value

return value type	block diagram icon	icon in "Tree Pane" and signature editor

scalar (logic, limited/ wrap-around integer, signed/ unsigned discrete, continuous)	 /0/scalar
	return/scalar
	/0/matrix
	return/matrix
	/0/enum_m
	return/enum_m
	/0/complex
	return/complex
	/0/array
	return/array
		return::cont

enumeration		return::Enumeration

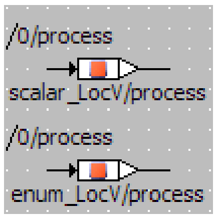
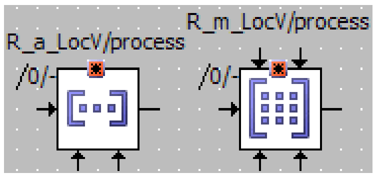
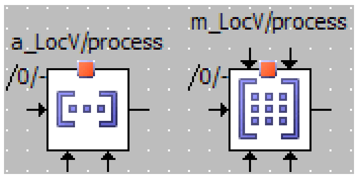
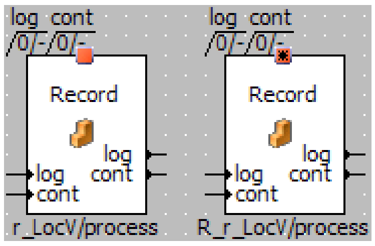
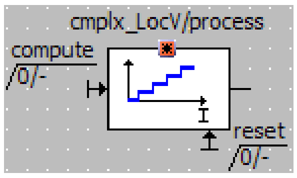
composite		return::array[cont]
		return::mat[cont]

complex		return::CountDown

5.5.2 Process Signature Elements

The signature elements of a process can only be created in the signature editor. A process can have the following signature elements:

- local variables

variable type	block diagram icon	icon in "Tree Pane" and signature editor
scalar (logic, limited/ wrap-around integer, signed/ unsigned discrete, continuous)	 /0/process scalar_LocV/process /0/process enum_LocV/process	scalar_LocV::cont enum_LocV::Enumeration
enumeration		
composite (explicit reference)		R_a_LocV::array[cont] R_m_LocV::mat[log]
composite (instance)		a_LocV::array[cont] m_LocV::mat[cont]
record (instance / explicit reference)	 log cont /0/-/0/- Record log log cont cont r_LocV/process log cont /0/-/0/- Record log log cont cont R_r_LocV/process	r_LocV::Record R_r_LocV::Record
complex (except record; always explicit reference)		cmplx_LocV::Counter

5.6 Miscellaneous Elements

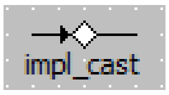
Several other elements are available in ASCET block diagrams:

- implementation casts (section 5.6.1 on page 32)
- hierarchies (section 5.6.2 on page 32)
- self (section 5.6.3 on page 33)
- comments (section 5.6.4 on page 33)

5.6.1 Implementation Casts

Implementation casts provide the user with the ability to influence the implementation of intermediate results within arithmetic chains. This allows the user to display knowledge regarding particular physical correlations (for example, that a specific range of values is not exceeded at a defined point in the model) in the model, without requiring the allocation of physical memory.

Implementation casts cannot be used in conjunction with logical elements.

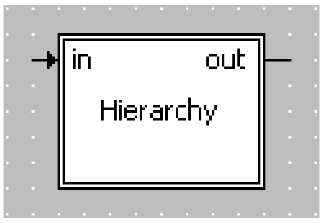
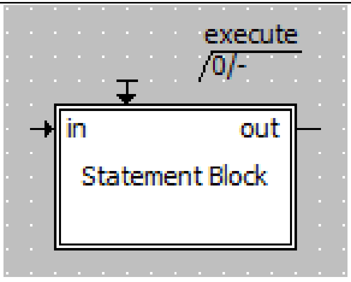
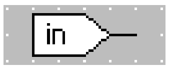
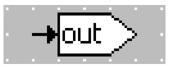
name	palette / toolbar icon	block diagram icon	"Tree Pane" icon
Implementation Cast			impl_cast::cont

5.6.2 Hierarchies and Statement Blocks

In order to structure a block diagram, *graphical hierarchies* can be used. Graphical hierarchies do not influence the semantics of a block diagram, but are used for structuring only.

Statement blocks can be used to encapsulate a continuous set of block diagram statements.

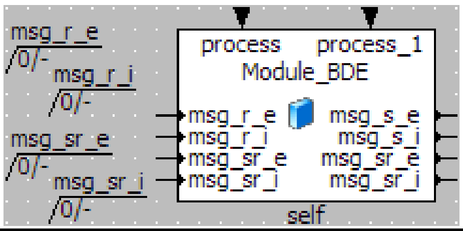
A hierarchy or a statement block contains a part of the block diagram. At its parent level of the diagram, it is visible only as a symbol. The lines that cross the border of the hierarchy or statement block, i.e. that connect elements inside the hierarchy/statement block with those outside, are represented by pins.

name	palette / toolbar icon	block diagram icon	"Navigation" tab icon[a]
Hierarchy			Hierarchy
Statement block			Statement Block
Inpin	*(none)*		in
Outpin	*(none)*		out

a. Hierarchies, statement blocks, their inpins and outpins are not shown in the "Outline" tab.

5.6.3 Self

The *Self* element is a reference to the currently edited component itself.

name	palette / toolbar icon	block diagram icon (= layout of currently edited component)	"Navigation" tab icon[a,b]
Self		 msg_r_e /0/- msg_r_i /0/- msg_sr_e /0/- msg_sr_i /0/- process process_1 Module_BDE msg_r_e msg_r_i msg_sr_e msg_sr_i msg_s_e msg_s_i msg_sr_e msg_sr_i self	self

a. The Self element is not shown in the "Outline" tab.

b. The icon depends on the component type (see also section 5.3)

5.6.4 Comments

ASCET block diagrams can include textual *comments*.

name	toolbar button	block diagram icon (= comment text)	"Navigation" tab icon[a]
Comment		This is a comment.	Comment

a. Comments are not shown in the "Outline" tab.

6 Miscellaneous Icons

This chapter lists various icons used for different purposes in block diagrams.

6.1 Scope Icons

The following scopes are available for ASCET elements:

- imported, exported, local, method-/process-local

Each scope is represented by a pattern on the basic variable or parameter icon.

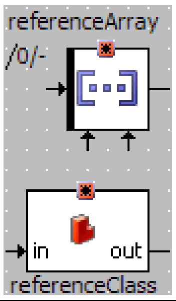
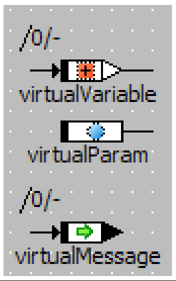
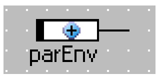
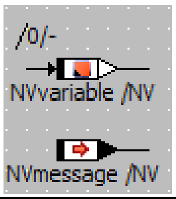
scope	definition	icon
exported	Exported elements are defined in one component and can be accessed by all other components by importing that element.	◩ ◓
imported	Imported elements are defined in another component or project, but can be used in the component that imports them.	◪ ◒
local	Local elements can only be used within the component that defines them, i.e. in all methods or processes of that component.	🟥 🔵
method-/ process-local[a]	Method-/process-local elements can only be used in the method/process that define them.	🟥

a. variables only

The scope icons appear in various places, e.g., the "Tree Pane" or the block diagram.

6.2 Icons for Properties

Several element properties are marked by overlay icons to the basic variable or parameter icon.

property	definition	block diagram icon	"Tree Pane" icon
reference[a]	Marks a reference type (i.e. composite or complex element) as explicit reference.		🟥
virtual[b]	Virtual variables/parameters are only available in the specification platform, they bear no relevance for code generation.		🟥 🔵
dependent[c]	Model parameters can be connected to other system or model parameters via a mathematical dependency.		🔵
non-volatile[d,e]	In the ECU, the element is placed in the non-volatile memory.		🟥

a. only available for variables

b. Virtual messages can be created, but they are not marked with an overlay icon.

c. only available for parameters

d. overlay icon only shown for non-volatile variables

e. Non-volatile messages can be created, but they are not marked with an overlay icon.

6.3 Icons in the "Tree Pane"

Most icons that appear in the tabs of the "Tree Pane" have already been mentioned. The remaining ones are listed here.

location	icon	explanation
"Outline" tab		*public diagram*[a], currently not loaded (left) / loaded (right, blue rim)
		private diagram[a], currently not loaded (left) / loaded (right, blue rim)
		additional marker for the currently loaded diagram
		method[a], public[b] (left) or private (right)
		process[a,c] (always public)
	#	indicates the process/method most recently used
"Navigation" tab		root node for the "Graphic Blocks" tree structure
	#	root node for the "Sequence Calls" tree structure
		connection line[d] between two diagram items

a. also used in the "Navigation" tab

b. A similar icon marks the "Methods" tab in the "Browse" view (cf. section 6.5).

c. in modules only

d. By default, no connections are shown in the "Navigation" tab.

Each tab in the "Tree Pane" contains some buttons.

button	icon	remarks
Change filter settings		"Outline" and "Navigation" tab only; can be used to filter the tabs. An active filter is indicated by a green overlay icon.
Change sort criteria		"Outline" tab only; can be used to sort the tab.
Expand All		
Collapse All		
Search the Tree		

You can customize the content of the "Outline" and "Navigation" tabs in the ASCET options window, "Appearance\Tree Pane\Filter\Navigation Tree" and "Appearance\Tree Pane\Filter\Outline Tree" nodes.

6.4 Icons in the "General" Toolbar

Besides default buttons such as **Save** or **Print**, the "General" toolbar contains the following buttons:

button name	icon	explanation	remarks
Switch to Connection Mode		starts the element connection mode	
Redraw		reloads the current diagram	
Save, Print, Cut, Copy, Paste, Delete, Undo, Redo			default functionality
Edit Component Data		opens the data editor for the edited component	A similar icon (without pencil) marks the "Data" tab in the "Browse" view.
Edit Component Implementation		opens the implementation editor for the edited component	A similar icon (without pencil) marks the "Implementation" tab in the "Browse" view.
Edit Default Project		opens the default project for the edited component	
Tool Options		opens the ASCET options window	
Insert Component		inserts a component as complex element	
Insert Method		creates a new method in the current diagram	A similar icon (without +) marks the "Methods" tab in the "Browse" view.
Insert Process		creates a new method in the current diagram	
Browse to Parent Component		opens an editor for the including component	Only available if the edited component was opened from the including component.
Generate Code		generates code for the edited component	
Compile generated Code		compiles generated code for the edited component	
Open Experiment		generates and compiles code for the edited component and starts the offline experiment	

button name	icon	explanation	remarks
Set Zoom to Page	🔍	sets the zoom factor so that the entire first page is shown	
Set Zoom to 100%	🔍	sets the zoom factor to 100%	
Set Zoom to Fit	🔍	sets the zoom factor so that the entire diagram is shown	

6.5 Icons in Tab Labels

The following table lists the icons used in the labels of the various tabs of the block diagram editor.

location	icon	remarks
"Outline" tab	Outline	These tabs are shown in the "Tree Pane", at the left of the editor window.
"Navigation" tab	Navigation	
"Database" tab *or* "Workspace" tab	Database Workspace	Name and icon of the third tab depend on whether you are working with a database or a workspace.
"Specification" tab[a]	Specification	These tabs are displayed vertically, near the right edge of the editor window.
"Browse" tab	Browse	
"Elements" tab	Elements	These tabs are subtabs of the "Browse" tab.
"Data" tab	Data	
"Implementation" tab	Implementation	
"Methods" tab	Methods	
"Layout" tab	Layout	

a. The icon depends on the type of the edited component (see section 5.3)

7 Contact Information

Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website: www.etas.com/hotlines

ETAS Headquarters

ETAS GmbH

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

Index