ICONNECT

Move our way

MICRO-EPSILON

SOFTWARE AND SYSTEMS

iCONNECT

Graphic development system
for the object-oriented design
of data processing-algorithms

Contents

Windows, Windows NT are trademarks of the Microsoft Corporation.

MICRO-EPSILON MESSTECHNIK
GmbH & Co.KG
Department Software
Königbacher Str. 15
94496 Ortenburg

Tel. +49/8542/168-0
Fax +49/8542/168-90
e-mail: info@micro-epsilon.de
http://www.micro-epsilon.com

Introduction

ICONNECT is a development and runtime system for signal processing tasks in the automation, measuring, and testing technology, and for process monitoring and process control. In a graphic environment the user can generate a signal graph that describes the data flow of a complex sequence. In a library ICONNECT provides algorithms (modules) that you “connect” on the screen using the mouse.

This tutorial has been written in order to enable the developer to design applications himself. The individual contents are explained with the help of examples that are all stored on the CD. The CD also contains an overview of the modules used in examples demo_01 to demo_47 (path: \manual\pdf\english\verzeichnis.pdf). The chapter following this introduction describes the definition of an application consisting of a signal graph without a user interface of its own. It also discusses the functions of the editor and demonstrates how individual applications can be documented in a comfortable way.

The graphical user interface allows to use the application without having detailed knowledge of the signal graph. You are able to integrate your own algorithm modules (see module programming manual) as DLLs with suitable interface in the development environment.

ICONNECT is a constantly growing software packet that is an ideal tool especially development of systems.

The online-help offers additional information about inputs/outputs, function, parameters etc. of the individual modules.

We wish you every success with your first steps!

Your MICRO-EPSILON team

Licence Agreement

Your right to use ICONNECT is subject to the terms in this licence agreement. Please read this carefully. If you do not agree to these terms, return the data medium package and all other items complete and unused to your authorized ICONNECT dealer for a full refund.

Warning!
It is illegal to make copies of ICONNECT without the prior consent of MICRO-EPSILON. Making copies will render you liable to pay damages and is an offence under §§106ff of the copyright law.

Warning!
Possession or use for commercial purposes of any program, device or other means intended to facilitate removal or circumvention of the hardware lock or node indentification number with which this copy of ICONNECT is supplied is a violaton of law.

1. LICENCE

MICRO-EPSILON grants you a non-transferable licence for using this copy of ICONNECT and the accompanying documentation on equipment owned by you or under your control according the terms below:

A. SINGLE-USER INSTALLATION

If you have acquired a licence for an ICONNECT single-user version, your copy may be used only as a single-user installation subject to the following conditions:

You may not allow this copy of ICONNECT to be used by more than one person or on more than one stand-alone computer or workstation at any time.

B. ADDITIONAL RIGHTS AND OBLIGATIONS

You may make copies of the original data media as archival copies to be used solely for backup. You may not copy ICONNECT or accompanying documentation except as permitted by this agreement. Any other copies of the whole or any part of ICONNECT are unlawful. You may not modify, translate or adapt ICONNECT or accompanying documentation, nor arrange or create derivative works based on ICONNECT for any purpose. The transfer of the right granted to use this product to a third person is permissible, but only if (a) all the data media together with the accompanying material are sold to the third person, (b) you do not keep any copies of ICONNECT (including copies on the hard disk), and (c) the third person explicitly undertakes to observe all the conditions of this licence. Your right to use this product becomes void with such a transfer.

The following applies to customers who are resellers:
Ultimate customers may only be granted a right to use this product, if they undertake to observe the regulations contained in this licence agreement.

It is not allowed to remove notes, labels, or trademarks of MICRO-EPSILON from your copy of ICONNECT or from the accompanying material.

The ICONNECT program may not be reconstructed, decompiled, or disassembled. If your copy of the ICONNECT program is equipped with a copy protection facility (hardware lock, dongle), this copy may only be used with this facility or with a replacement facility supplied by MICRO-EPSILON or by an authorized ICONNECT dealer. It is not allowed to remove or avoid the copy protection facility.

2. OWNERSHIP

MICRO-EPSILON remains the holder of copyrights and other protective rights in ICONNECT and the accompanying material, also if copies have been made thereof with the consent of MICRO-EPSILON. The software and the documentation contain business secrets of MICRO-EPSILON and/or their licenser; they are copyrighted. The customer will observe this, and will especially not delete copyright notes and/or serialisation numbers.

The customer will not make the software and the documentation available to third persons who are not resellers or ultimate customers, without the written consent of MICRO-EPSILON.

3. UNAUTHORISED COPYING

If you copy the ICONNECT program or the accompanying documentation without permission, or if you violate any other regulations of this licence agreement, your right to use the product will be terminated immediately. In this case MICRO-EPSILON reserves all rights.

4. WARRANTY

MICRO-EPSILON warrants that ICONNECT will perform substantially as described in the ICONNECT Reference Manual and the data media on which ICONNECT is furnished will be free from defects. Excluding any other liability MICRO-EPSILON or the authorised ICONNECT dealer are prepared - according to our choice - within a period of six months starting from the moment the program package is delivered, to either (a) establish proper usability of the program with suitable effort and within an adequate time, (b) to replace your copy of ICONNECT by functionally equivalent programs, or (c) to pay back the purchase price against return of the product, which means that the licence will expire.

There is no further warranty. Especially there is no warranty that ICONNECT will satisfy your special requirements. You are solely responsible for the selection, installation, and for the use of the program, as well as for the results intended to be achieved with the program. MICRO-EPSILON neither guarantees uninterrupted or error-free operation of the program, nor its suitability for a certain purpose.

5. LIMITATION OF LIABILITY

If there should be stringent regulations demanding a liability on MICRO-EPSILON's side, such liability will be limited to culpably caused and predictable damage and to an amount only as high as the sales price. Any liability for lost profit, occurred losses, indirect damage, and consequential damage is excluded. Liability for loss of data is also excluded. The user acknowledges that this distribution of the risk is adequate in view of the amount of the licence fee. These limitations of liability do not apply to assured properties and to damage that is due to intent or gross negligence of MICRO-EPSILON, or which result from the violation of such contractual obligations that are of basic importance for the establishment of the purpose of the contract.

6. GENERAL REMARKS

This agreement is not governed by the standard U.N. Convention on Contracts for the International Sale of Goods, but only by the law of the Federal Republic of Germany. Any disputes that might arise from this licence agreement will be subject to German jurisdiction. The place of jurisdiction will be Passau. This licence will expire without further notice or action from MICRO-EPSILON, if bankruptcy or insolvency proceedings are instituted against you, or if your company is dissolved. This agreement reflects all the possible arrangements with MICRO-EPSILON and replaces any other declarations of intention and advertising messages referring to the ICONNECT product and to the accompanying material. There have not been any oral collateral agreements. The contractual regulations will remain effective in their remaining parts, even if individual regulations should be ineffective. Ineffective regulations are considered to be replaced by such regulations that come as close as possible to the desired economic success in a legally permissible way. „MICRO-EPSILON“ and „ICONNECT“ are registered trademarks of MICRO-EPSILON MESSTECHNIK GmbH & Co. KG.

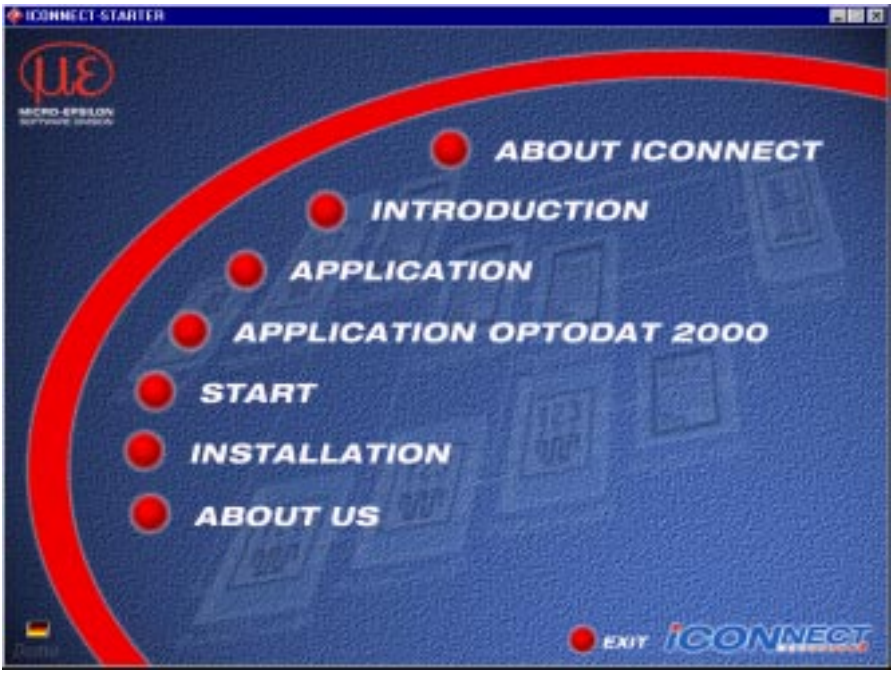
1. Installation

Please read the licence agreement carefully before starting the installation!

The full version of ICONNECT is installed as follows:

1. Plug the supplied dongle into the parallel port LPT1 of your computer.

2. Insert the CD into your computer's CD-ROM.



Pic. 1.1: Start screen

Select **Installation** from the main menu (see pic. 1.1) and then **ICONNECT Complete Install**.

3. Follow the on-screen instructions and select the ICONNECT installation directory.

4. After installation licensing must be done. Insert the licence disk into your floppy disk drive and press the **Start** button.

The **demo version** of ICONNECT is installed as follows:

1. Insert the CD into your computer's CD-ROM drive. Select **Installation** from the main menu (see pic. 1.1) and then **ICONNECT Complete Install**.

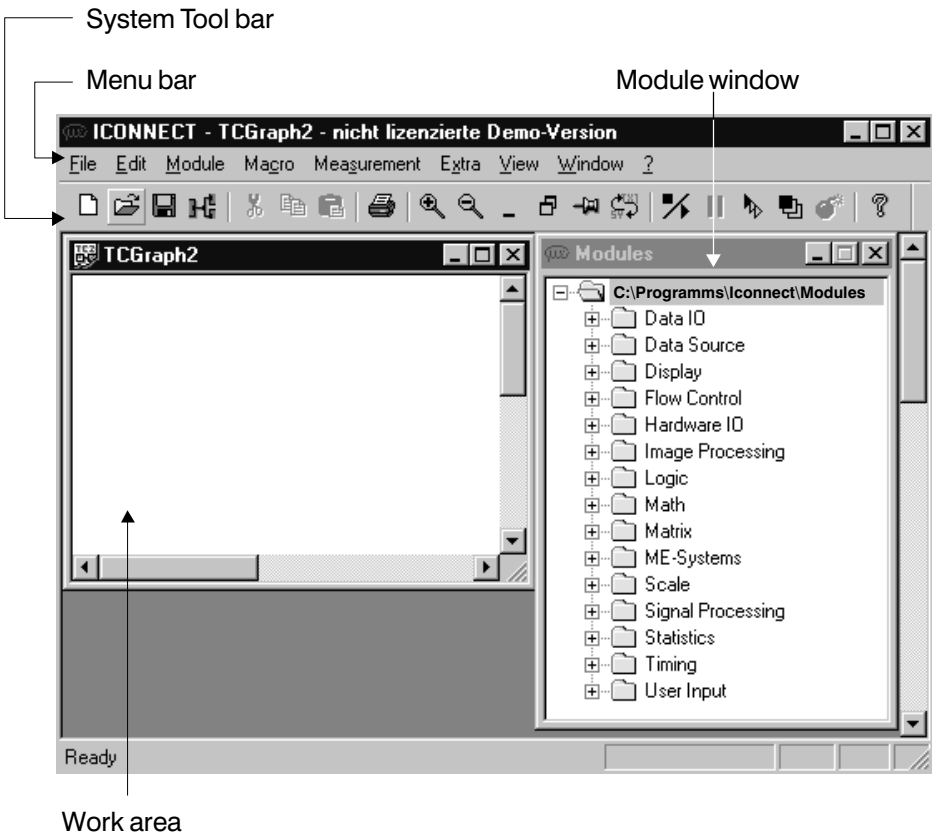
2. Follow the on-screen instructions and select the ICONNECT installation directory.

3. The demo version of ICONNECT needs no licensing. Press the **Cancel** button to close the licensing dialog.

2. Editor Functions

When ICONNECT is started the log-in dialog is shown. Repeat the user name (default value) for the password and confirm with the OK button. If you use the demo version, please click on the OK button without entering a password. Arrange the icon bars (see pic. 2.1a) with pressed left mouse button similar to pic. 2.1b.

Pic. 2.1a: Dragging the tool bar



Pic. 2.1b: ICONNECT development environment

The **menu bar** allows you to access editor functions, control functions for the signal graphs, and user administration functions (see chapter 5).

The **system tool bar** provides access to the most important ICONNECT functions by clicking on tool buttons.

The **module window** allows the selection of modules.

The **work area** is where the signal graph is constructed with modules.

The example shown in pic. 2.2 should demonstrate the capabilities of the development environment.

Please note:
In the following chapters left-click means pressing the left mouse button once, right-click pressing the right mouse button once. Double-click-left means clicking the left mouse button twice. Double-click-right is similar.

2.1 Drawing Signal Graphs

Applications are generated by drawing signal graphs. This chapter demonstrates the fundamentals for this process. The parameterisation of a function generator and the presentation of its output signal on a display will be used as an example.

2.1.1 The Signal Graph, the ICONNECT Program

Pic. 2.2 shows ICONNECT with open windows on the screen. The window named **demo_01 - Active signal graph!!!** shows an example of a signal graph. The individual modules perform different tasks, such as for example I/O operations, logic and arithmetic functions, and visualisation of processed measurement data. All the modules have their inputs on the left side and their outputs on the right side. By means of these module ports the data are supplied to the parameterised algorithm and are output after processing. The window named **AnalogChart1** is the display window that belongs to the module with the same name. This window displayes the scaled output of the module SinGen1. The signal graph is controlled by means of the

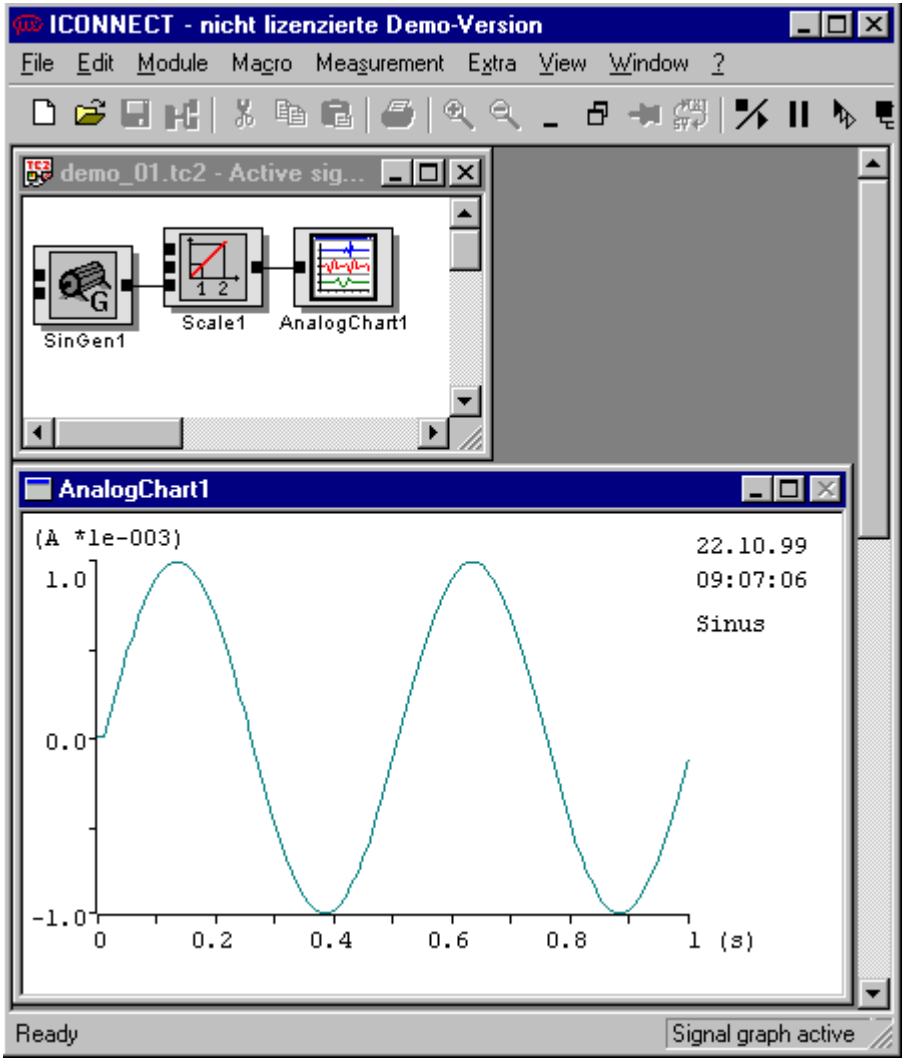
 Start / Stop

 Pause

icons in the system tool bar or by using the menu. The corresponding items in the **Measurement** menu are **Start**/**Stop** and **Pause**.

The function of a signal graph can be recognised from the block diagram. In the signal graph of pic. 2.2 a sine-shaped signal is generated in the **SinGen1** (FuncGen) module. The **Scale1** module scales this signal to a range of -1 to 1 milliampere. ICONNECT carries information about the characteristic properties of a signal in all the communication channels. This information also contains the unit and the range of a signal. In every module this information is passed through or modified. This guarantees that the displays show the correct units (see chapter 4). In **AnalogChart1** the recorded data are, in accordance with the parameter settings, sent to the display window, where they are displayed then.

The realisation of this example will be discussed below, with special emphasis on the individual functions of the editor and the help system.



Pic. 2.2: ICONNECT program

Please note:
All the examples discussed or presented in this tutorial are stored in the directory c:\examples\demo, c being the drive letter of your hard disk.

2.1.2 Selecting Modules

The following steps have to be performed in order to select the function generator:

1. Left-click on menu item **Module**

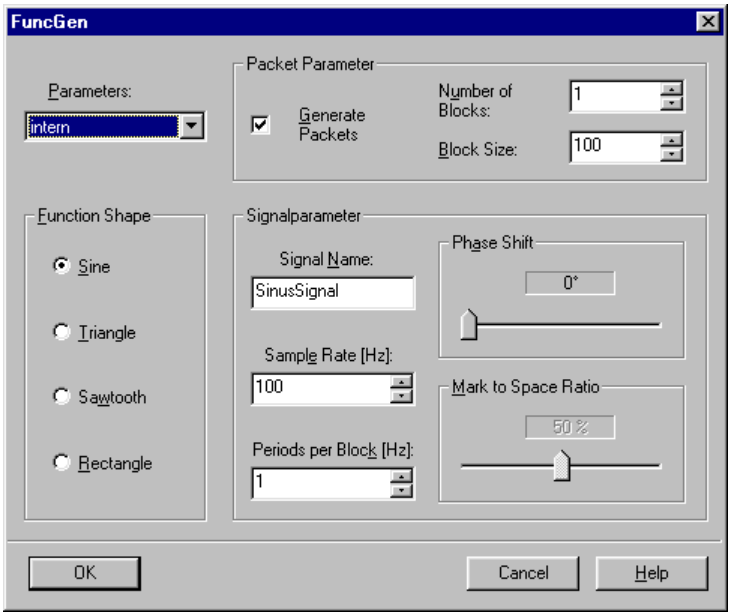
2. Move the mouse pointer down to the **User INPUT** submenu, then right to **FuncGen** and left-click again. The icon representing the algorithm appears in the work area.

3. The icon can be moved across the work area with the mouse, and it can be placed at the desired position with a left-click.

The two other modules **Scale** (**Module** > **Scale**) and **AnalogChart** (**Module** > **Display**) are selected and placed in the same way.

2.1.3 Defining Module Parameters

In general a module has its module-specific dialog. There are two ways of opening this dialog: Either with a double-click-left on the symbol of a module icon, or with a right-click on the module icon, which will open a pop-up menu where the item **Properties** must be selected with a left-click.



*Pic. 2.3: Dialog of the **FuncGen** module*

The FuncGen module generates signals like sine, rectangle, triangle, and saw-tooth. See the online help for additional information.

The packet parameters **Block Size** and **Number of Blocks** characterise the data flow between the modules. ICONNECT supports block-oriented communication. Data belonging to a defined unit are gathered to form a packet. A packet represents a number of data that belongs together semantically, e.g. a picture, a measurement, or the data of a certain workpiece. The packet is provided with information (see **TypeInfo** chapter 4).

A packet comprises at least one data block, which in turn consists of at least one data item (e.g. a measuring point). This two-step hierarchy of packets and blocks makes sense, because it allows the calculation of intermediate results in packets that are very large or are measured over a long period of time. When a signal graph is developed it must be taken into consideration that some modules only work in mode packet or block. The **Generate Packets** checkbox can be used to determine whether a packet with the corresponding number of blocks is generated as an output signal or if this is not selected whether a continuously running packet i.e. a permanent measurement is generated.

2.1.4 Selecting Parameter Sources in ICONNECT

Parameters can be generated in three different ways:

1. Parameters that are defined in the dialog are called **internal** parameters.

2. If database is selected as parameter source, the module port **DB** can be used in the example with the function generator for reading in the frequency and the sampling rate during the application runtime from a database.

3. With the **EXT** input any data from the signal graph can be used for the corresponding parameters, if **external** is activated as parameter source (see chapter 3.2.2).

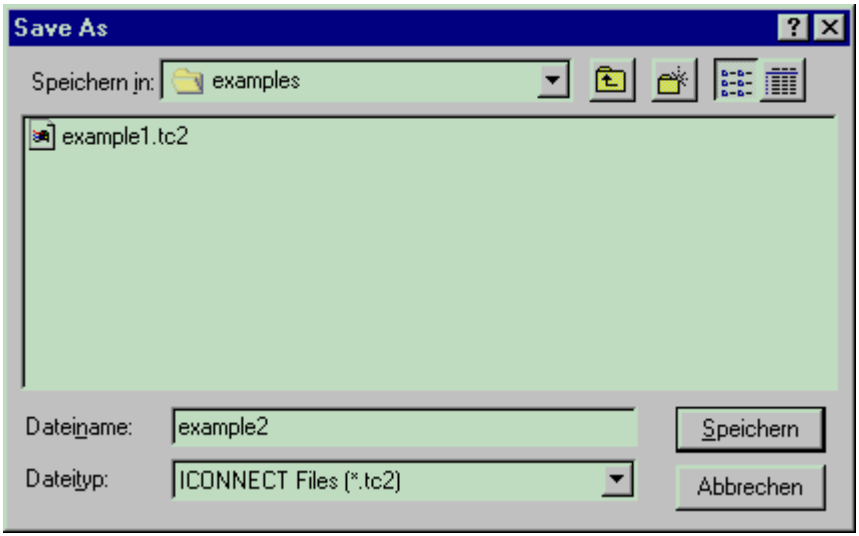
Please note:
*Chapter 4 contains more information about **Block Size**, **Number of Blocks** and **Packet**.*

2.1.5 Connecting Module Ports

If the mouse pointer touches a module port, the name of the input or output will be displayed. A left-click on the module port generates the start of a communication channel. A left-click on the port of the corresponding second module completes the connection. A connection that has been started by mistake can be removed again with a left-click in the work area. ICONNECT will “tidy up” your work area (rearrange the module icons and connect them with straight lines), if you select **Right angled wiring** in the **Edit** menu.

2.1.6 Saving and Loading Signal Graphs

The WINDOWS file dialog is opened with a left-click on the **File** menu and then by choosing **Save as**. The suffix **.tc2** is automatically added to the file name that is entered in the respective line. The file is then saved with a left-click on the **Save** button.



*Pic. 2.4: Dialog of the **Save as** menu item*

A signal graph can be loaded by choosing **Open** in the **File** menu, or by selecting the **folder icon**. Type in the information and confirme. The actions may also be performed with the **keyboard-shortcuts** that are mentioned in the menus.

2.2 Online Help

The online help is started by selecting the **question mark** in the menu bar and then by selecting the sub-item **Contents**. Apart from the above-mentioned possibility information about the modules can also be obtained with a right-click on the module icon and then by selecting the help item.

2.3 Editing the Modules in the Signal Graph

Delete

Select the module to be deleted with a left-click, then press the **del**-key on the keyboard. As an alternative an object can also be deleted with the menu item **Delete** in the **Edit** menu.

Cut, Copy, Paste

Select the module and choose one of three methods known in Windows.

- **Edit** menu
- Pressing the buttons **scissors**, **double document**, and **envelope**
- Entering the corresponding **keyboard-shortcuts** that are listed in the **Edit** menu (e.g. **Strg X** for cut)

Move a module

Select the module and move it with pressed left mouse button. The module is positioned when the mouse button is released.

Please note:
*The module is referenced by its name and a modules instance (e.g. **Scale1**, pic. 2.2). In an ICONNECT session the module numbers are incremented. It is therefore possible that there are differences between the signal graphs shown in this tutorial and the graphs on your screen.*

2.4 Editing a Module Group in the Signal Graph

The actions described in chapter 2.3. also apply to a group of modules. Several modules can be selected by

- selecting the objects with left-click and pressed Shift key.
- drawing a selection frame. For this purpose keep the left mouse button pressed down in the work area and use the mouse to draw a frame across the modules.

In the **Edit** menu you may use the item **Select All** to select the complete signal graph. The selection of a group can be undone with a left-click in the work area.

2.5 Example

In this example you will create a random number generator. One value will be generated every second, and it will be visualized in a digital display. The completed signal graph is stored in the path \examples\demo\demo_02. We recommend that you create the program yourself. For this purpose please perform the following steps:

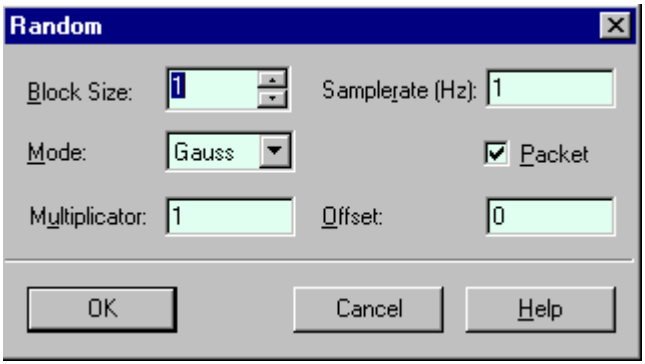
1. Select and place the **Random** module by using the menu **Modules > User INPUT > Random**, or by using the respective icon from the **User INPUT** tool bar. When an icon is selected with the mouse pointer, its name is displayed in a tool-tip.

2. Double-click-left on the module icon to open the parameter dialog (see pic. 2.5). Setting the parameters block size = 1, sampling rate = 1 Hz, and Gauss distribution, will generate a random number every second. As already mentioned the **packet** parameter determines the characterisation of the output data. In this example packets containing one data block with one value each will be transmitted.

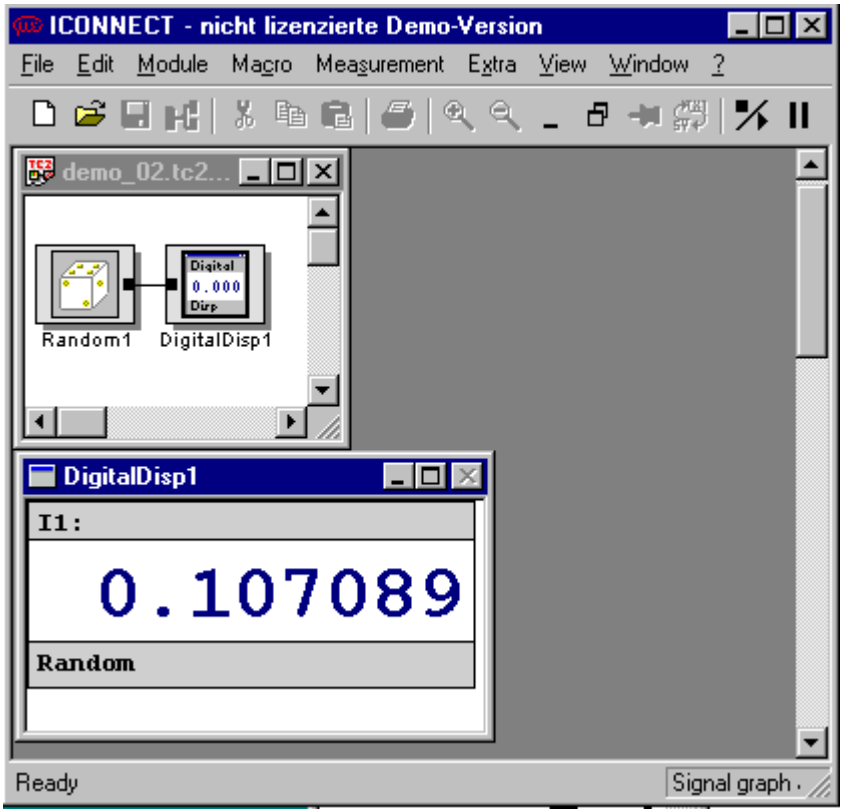
3. Select and place the **DigitalDisp** module (see chapter 2.1.2).

4. Connect the two ports (see chapter 2.1.5).

5. Activate the start symbol.



*Pic. 2.5: Dialog of the **Random** module*

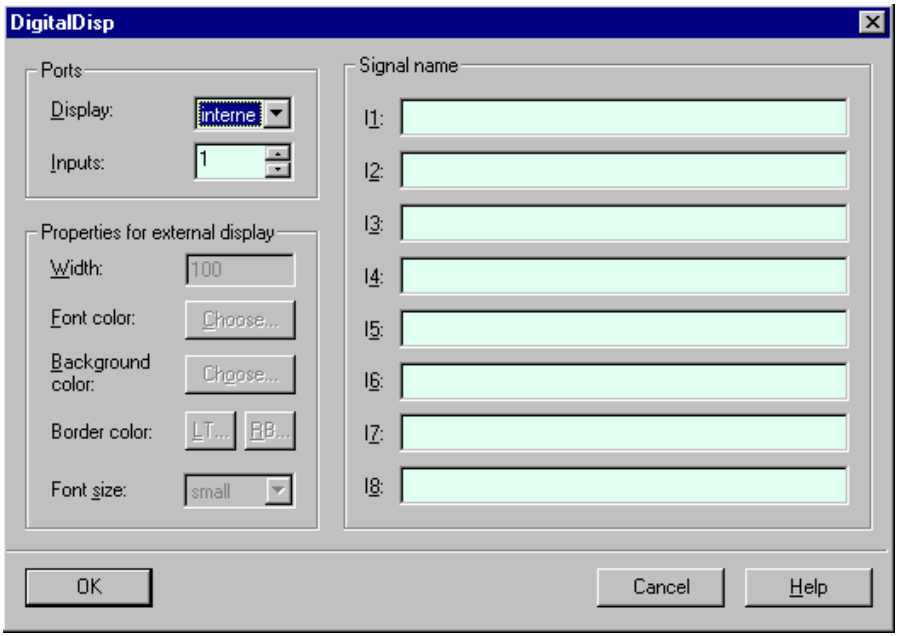


Pic. 2.6: Generation of a random number in ICONNECT

Then examine the possibilities of the digital display. For this purpose:

1. Open the **parameter dialog** of the **DigitalDisp** module with a double-click-left on the module icon (see pic. 2.7), or with a right-click and selecting the **Properties...** menu item.

A digital display can visualize up to eight signals in its window. The default value is 1, which is not changed in this example. You can define a signal name for each input. If you do not enter a name here, the display will show the name of the signals Type-Info (see chapter 4). With the random number generator this name is Random. If you do not want any text, please type in a blank space in the **signal name** field.



*Pic. 2.7: Dialog of the **DigitalDisp** module*

2. Type in the signal name **Signal1** at **I1**.

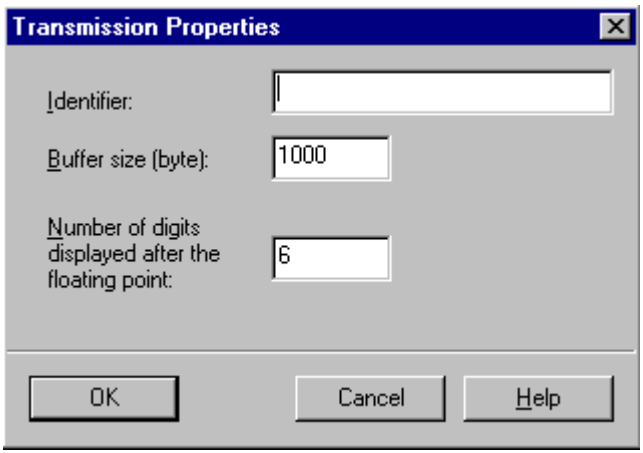
With the two buttons **Font color:** and **Background color:** you may set the font colour and the background colour with the WINDOWS colour dialog.

3. Set the font size to **Large**.

You will notice that due to the large number of decimal digits the value to be displayed does not completely fit into the window. Therefore modify the number of decimal digits.

4. Right-click on the wire between the two modules. Select the **Properties** item in the activated menu.

This action opens the properties dialog of the communication channel (see pic. 2.8)



Pic. 2.8: Properties of a communication channel

The buffer size determines the storage capacity of the communication channel. The default value for this parameter is 1,000 byte. If more storage capacity is required, the program reallocates the corresponding storage space. In case of large volumes of data it is therefore possible that the first pass of the signal graph takes a little longer, because it may be that the storage capacity for the communication channels must first be adapted. The accuracy of the communication channel, the default value of which is 6, determines the number of decimal digits.

5. Reduce the line accuracy to 3 decimal digits and confirm your entry. The changes will only become effective when the program is started.

Please note:
*In the **Extra > Standard Settings...** menu you may set the line accuracy as a global default for future communication channels.*

2.6 Summary

In this first chapter you learned about the functions of the graph editor in ICONNECT and about the fundamentals of generating simple signal graphs. The chapter discussed details in some modules, and the Type-Info contained in every data packet was mentioned, and the possibility of parameterization communication channels.

In the next chapter the module library of ICONNECT will be explained in several examples. The main emphasis will be placed on those modules that are used most frequently.

3. The ICONNECT Module Library

This chapter will extend your knowledge about signal graphs by introducing new modules from the ICONNECT module library. You will

- learn about **arithmetic modules** and their properties with respect to communication.
- discuss the modules of the **User Input** group, with the help you can interact with your ICONNECT application.
- generate user interfaces with the **DisplayManager** and **InputManager** modules

3.1 Arithmetic Modules

3.1.1 The VecOpVec Module

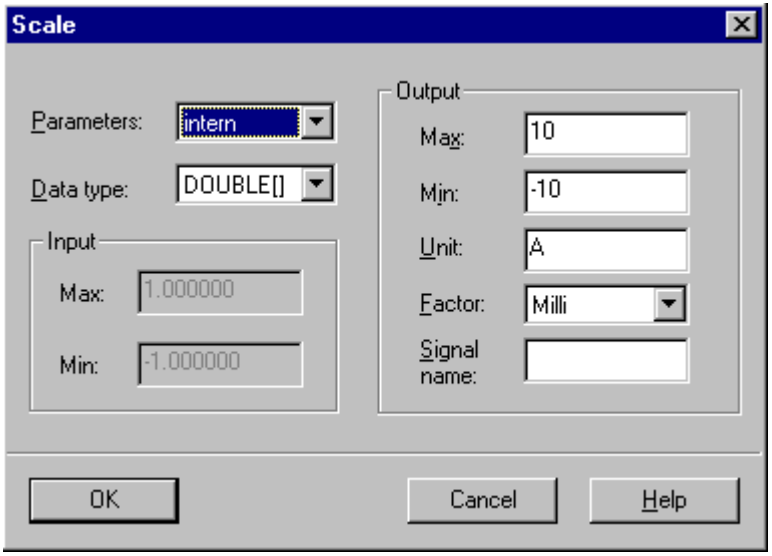
As an example you will add the signals of the modules **FuncGen** and **Random** (see pic. 3.5). The following steps are necessary for this purpose:

1. Place the **FuncGen** module (**Modules** > **DataSource**), and make the following entries in the dialog:
   Sample rate: 512 Hz
   Block size: 512 values
   Periods per block: 2 Hz

These settings will result in two periods per second with 256 values per period.

2. Insert the **Random** module into the work area.
   Parameters: Block size: 512 values
   Sample rate: 512 Hz
   Mode: Uniform
   Packet output

Uniform will generate uniform distributed values in the range between 0 and 1.



*Pic. 3.1: Parameters of the **Scale1** module*

3. Select the **Scale1** (**Modules** > **Scale**) module and assign the parameters as shown in pic. 3.1.

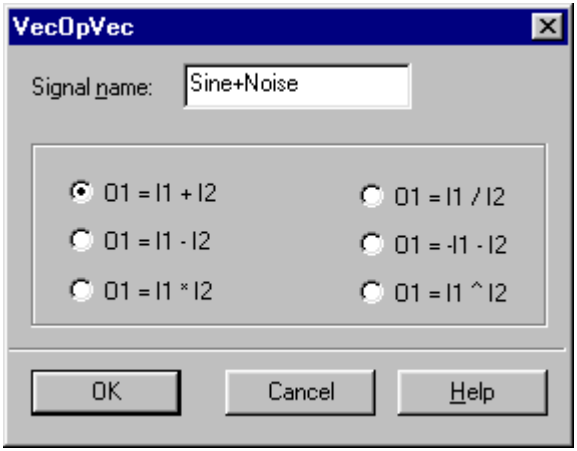
For the outputs -10 is entered as a **Min**-value, and 10 as a **Max**-value. This information determines the range of the signal. If **Type-Info** is selected in a display for the range of the ordinate to be shown, this range information will be used for scaling.
A for ampere is entered as the unit, and **Milli** ($=10^{-3}$) as the scaling factor. This unit, which also belongs to the Type-Info, is carried along over the complete signal graph and is calculated and checked in the arithmetic modules (see description of the “VecOpVec” module).
The parameter **DOUBLE[]**determines the data type for the module. The module expects a vector of real numbers of any length at the input. The type **DOUBLE** could be defined alternatively, which represents a single real number that is also called a real scalar.
With the help of these type reference the communication structure is already validated when modules are connected.

4. Select the **Scale2** module. The parameters are the same as for Scale1, except for the signal range, which shall be specified with -5 to 5.

5. Add the **VecOpVec** module from the **Module > Math** menu to the work area.

The **VecOpVec** module performs simple arithmetic operations with two real vectors. The addition operation requires the same physical units. This is implemented in steps 3 and 4, because the **VecOpVec** module checks the units for consistency. In case of additive operations different units will result in an error message. In case of multiplicative operations a corresponding new unit will be calculated. Identical block length and sampling rate are further prerequisites for a successful arithmetic operation with vectors.

6. Open the dialog of **VecOpVec** (see pic. 3.2), and change the signale name from **VecOpVec** to **Sine+Noise.**



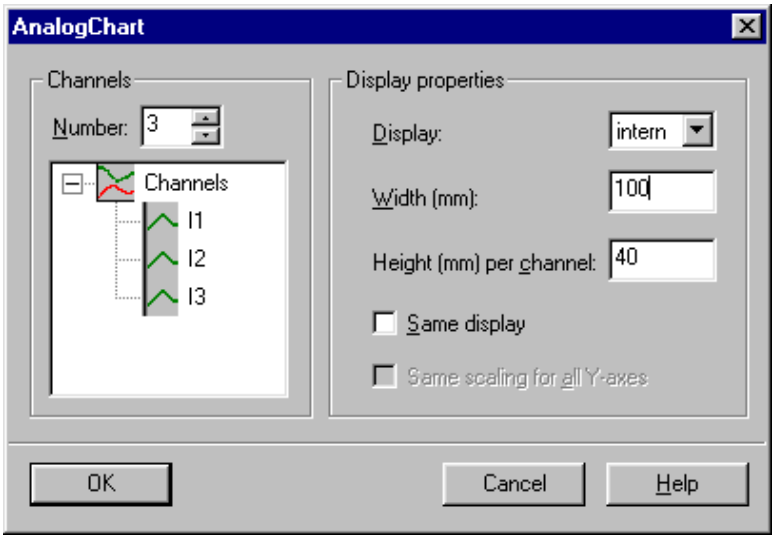
*Pic. 3.2: Dialog of the **VecOpVec** module*

7. Insert the **AnalogChart** module from the **Module > Display** menu into the work area.

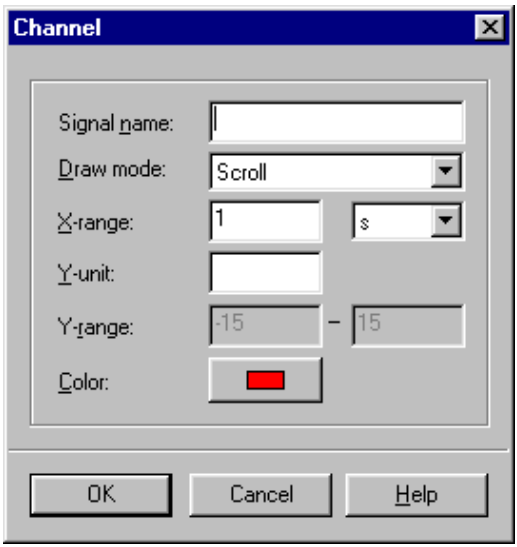
In this example parameterization shall be discussed in more detail. The dialog for this module is shown in pic. 3.3. The **Number** parameter in the **Channels** section defines three inputs, which are displayed in the tree-view below. A double-click-left on such an entry activates the submenu for channel setup (see pic. 3.4). Enter a value of 1 s for **the X-range**. The signal name and the y-axis notation are taken from the **Type-Info**. This is done automatically, if nothing is entered in the respective input lines. The colour of the graph can be set with the **Color** button.

8. Connect the modules acc. to pic. 3.5, and click on the start button.

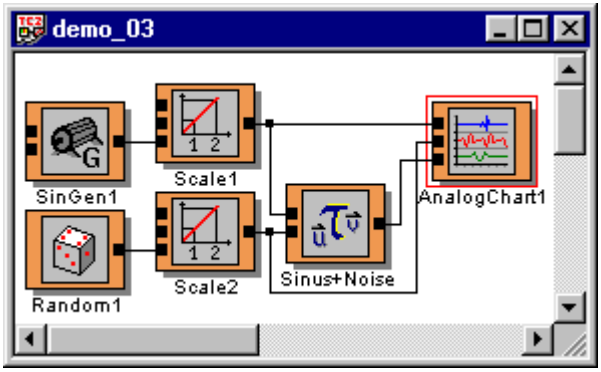
Pic. 3.6 shows the visualization of the individual signals. Apart from the units, value ranges, and signal references of the individual data flows the date and the time of signal generation are also displayed. This information that is referred to as a timestamp is also carried in the **Type-Info**.



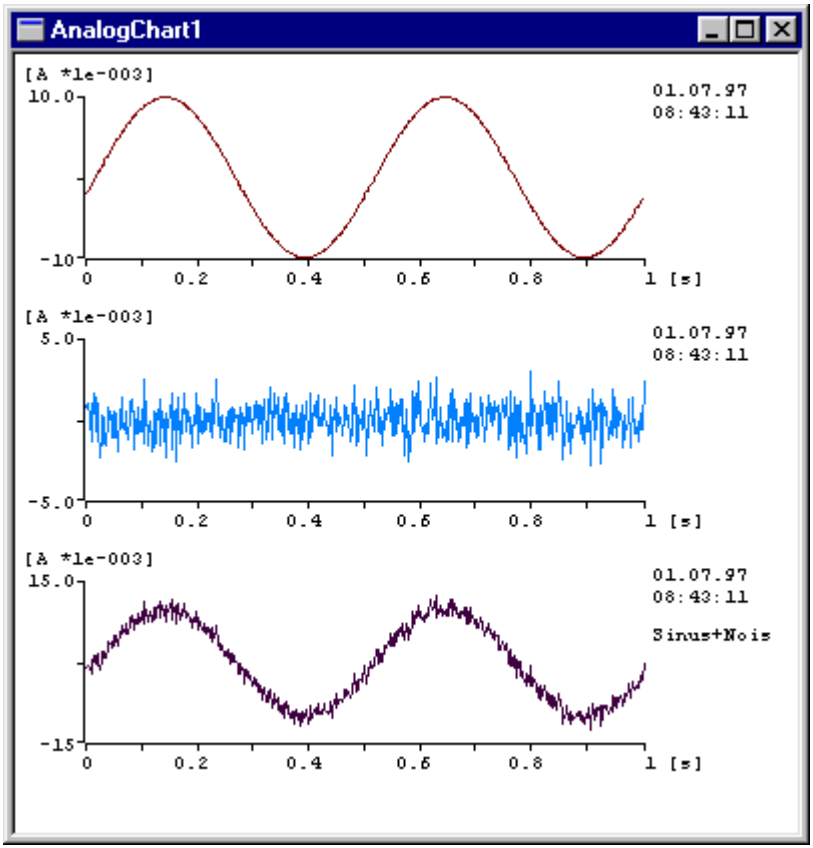
*Pic. 3.3: Dialog of the **AnalogChart** module*



*Pic. 3.4: Channel setup of the **AnalogChart** module*



*Pic. 3.5: Signal graph of **demo_03***



Pic. 3.6: Addition of sine signal with white noise

Please note:
*Select **multiplication** (01 = I1 * I2) as operation instead of **addition** in the **VecOpVec** module, and change the **unit** from **A** (ampere) to **V** (volt) in the **Scale2** module. Start the program again and observe the changes compared to pic. 3.6. In the bottom coordinate system **W**, i.e. **A*V**, is displayed as the **unit**.*

3.1.2 The VecOpScal Module

The above example introduced the **VecOpVec** module for arithmetic operations with two real vectors. In practice it is often necessary to combine a vector with a scalar, for example if in a measuring task a measurement signal should be combined with a correction value for thermal stabilization. The signal will be available as a vector, and the correction factor as a scalar. The **VecOpScal** module has the following properties:

- The **SCA** input takes in A scalar when the value of the scalar has changed.
- If there are data at the **VEC** input, these will be combined with the current scalar.
- If there are data at the **VEC** input without any data having been read at the SCA input at that time, a combination with the neutral element of this operation will be performed.

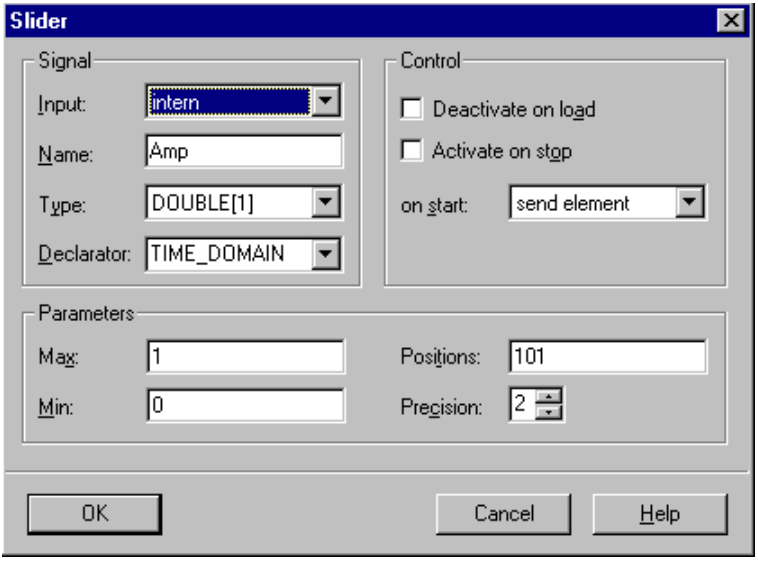
In the example (**demo_05**) the amplitude of the white noise can be adjusted interactively. The following steps are required for realising this program:

1. Place the **SliderV** module from the **Module > User Input** menu into the work area.

Modules from the **User INPUT** group that allow access to the signal graph during the application runtime will open a window with the corresponding Windows dialog element.

2. Type in **Amp** as the **Signal Name** in the dialog of the **Poti** module (see pic. 3.7), and with **DOUBLE[1]** define a vector of length 1 as the signal type.

3. Place a **Scale** module into the work area, and use this to scale the signal of the **Poti** module, which outputs values in the range from 0 to 1. Select **Milli** as the scaling factor. Do not enter a **unit**, because white noise will be multiplied with this signal. Due to the addition with the sine signal the resulting data flow must have the **unit** A.



*Pic. 3.7: Dialog of the **Poti** module*

4. Add the **VecOpScal** module from **Module > Math** to the signal graph. The dialog of this module is the same as in **VecOpVec**. Select the **multiplication** function as the **operation**, and type in **Amp** as a **signal name**.

5. Complete the program with the signal graph from example **demo_04**, and connect the modules with each other (see pic. 3.8).

A validation check for:

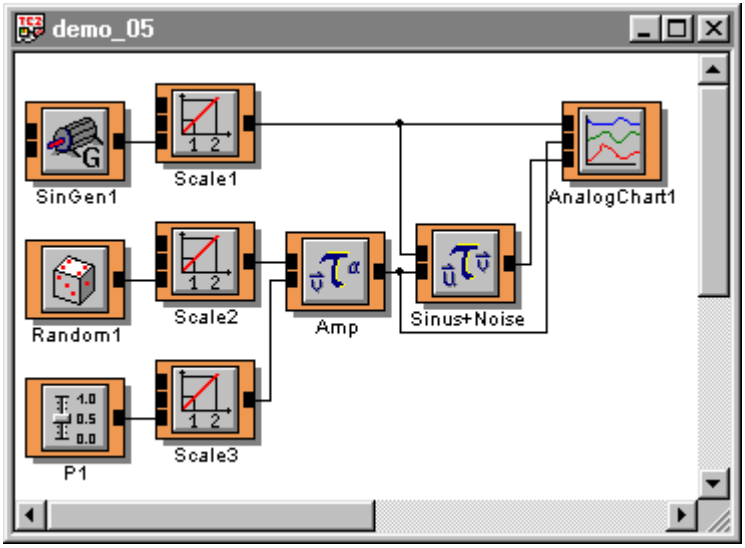
identical data types,
identical signal declarators

is performed when the modules **Scale3** and **VecOpScal** are connected.

If the check for identical declarators is not positive, an

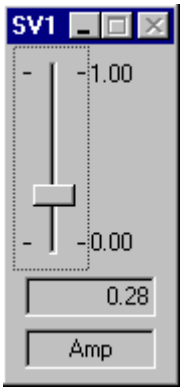
confirmation

will be performed. ICONNECT will issue a warning stating that the type declarators are not identical. This warning will be issued twice. In this example it has to be confirmed with **Yes**. This warning may be confirmed for every direction, if you agree with the declarators.



*Pic. 3.8: Signal graph of **demo_05***

5. Click on the start button. At the **Amp** control element (see pic. 3.9) you can adjust the amplitude of the white noise that is added to the sine signal.



*Pic. 3.9: Control element of the **Poti** module*

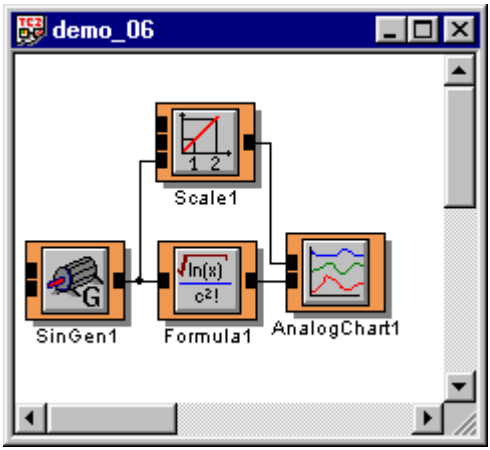
Please note:
When a connection is established between two modules the corresponding validation strings will be compared. For this purpose the graph editor sends the string of the source module to the target module. This target module compares the strings and confirms or negates the compatibility of the ports. Next the target module sends the string to the source module, which then performs the check. If this validation fails at one of the modules, the communication channel will not be created. Double-checking is necessary because in this check some modules adapt to the partner module with respect to their method of operation. Such modules may be both the source and the target module of a connection.

The validation string first contains the data type of the connection. With the ***Scale*** *module this is* ***DOUBLE[].*** *The* ***VecOpScal*** *module expects data of type* ***DOUBLE[1]*** *at the* ***SCA*** *input. These types are compatible with each other. At the application runtime the* ***VecOpScal*** *module detects whether* ***Scale*** *transmits more than one data value per block.*

In addition the signal to be transported on the generated communication channel is assigned a designation (= type designator or interpretation). The ***DOUBLE*** *type does not state whether the signal is a time, frequency, or control signal. The* ***Scale*** *module applies a time signal called* ***TIME_DOMAIN*** *at its output.* ***VecOpScal*** *expects a scalar, the interpretation of which is called* ***SCALAR****, at its input. If the type declarators are not equal, the checking module will issue a warning. If the user decides that the interpretations of the signals are compatible for his purposes, he will confirm the system' s question with* ***Yes****, and the communication channel will be created. Chapter 4 contains further details on data types and interpretations.*

3.1.3 The Formula Module

The next module from the **Math** group that will be introduced is the formula interpreter **Formula** in example **demo_06** (see pic. 3.10). In addition to the formula interpreter the modules **FuncGen**, **Scale**, and **AnalogChart** will also be used again. For these modules the same settings are used as in example **demo_5**.



*Pic. 3.10: Signal graph of **demo_06***

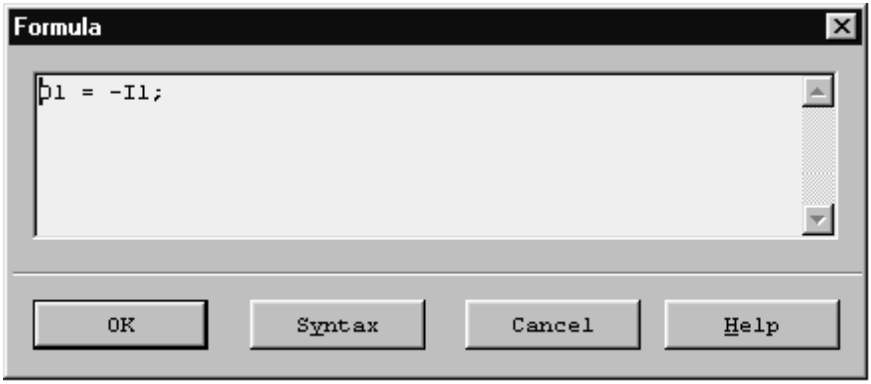
The dialog of the **Formula** module (see pic. 3.11) features an edit field for entering the formulas. A formula is structured as follows:

Module output = f(module input a, module input n, ...)

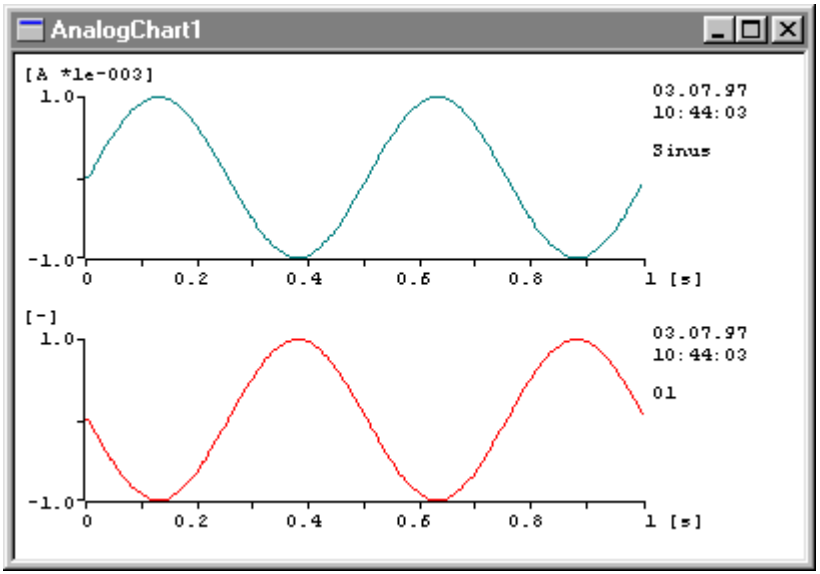
In a formula the variable at the left side of the equal sign is interpreted as the module output. The declarators at the right side of the equal sign are taken as inputs. In the example used the formula

01 = -I1;

has been entered. The sine signal present at the input is inverted.



*Pic. 3.11: Dialog of the **Formula** module*



Pic. 3.12: Inversion of a sine signal

The data types have to be entered for the individual variables in the formula (= inputs/outputs of the module), except if a variable has the default type (see tab. 3.1). The variable type determines the output type. It also decides whether the Formula module calculats and sends new data to the outputs or not. These actions are performed if there are data at the inputs that have a data type with trigger properties. If there are data at the inputs that do not trigger the calculation, the data will be taken into the module, but there will be no calculation.

The type designator consists of the letter **d** (data type = integer) or **f** (data type = floating-point), and a prefix. The prefix determines the characteristic (scalar or vector) and the trigger property.

Characteristic	Scalar				Vector			
Prefix	'&'		'%'		'#'			
Type designator	'&d'	'&f'	'%d'	'%f'	'#d'	'#f'	'd'	'f'
Data type	SWORD	DOUBLE	SWORD	DOUBLE	SWORD[1]	DOUBLE[1]	SWORD[]	DOUBLE[]
Trigger	YES	YES	NO	NO	NO	NO	YES	YES
								Default

*Tab. 3.1: Data types in the **Formula** module*

The type declarators are placed in square brackets after the variable name.

Examples:

Task: Define an input of type SWORD[1]
(= vector of integers with length 1).
Solution: ... = I1[#d]

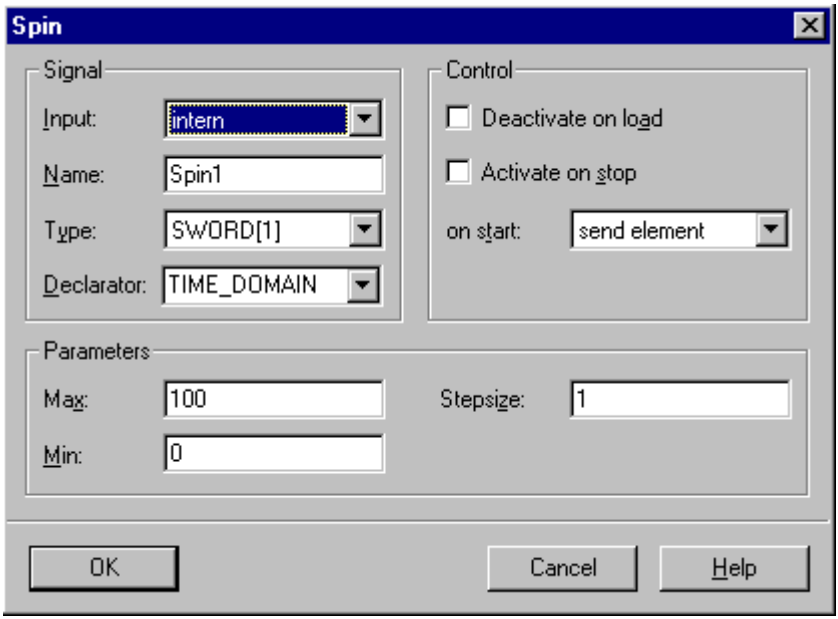
Task: Define an input of type DOUBLE
(= scalar of type floating-point without trigger property).
Solution: ... = I2[%f]

In the formula of pic. 3.11 no explicit type definition was set for the variables, because the default type was chosen. In the example **demo_07** the formula was extended to

```
01 = -(pow(I1,I2[%d]));
```

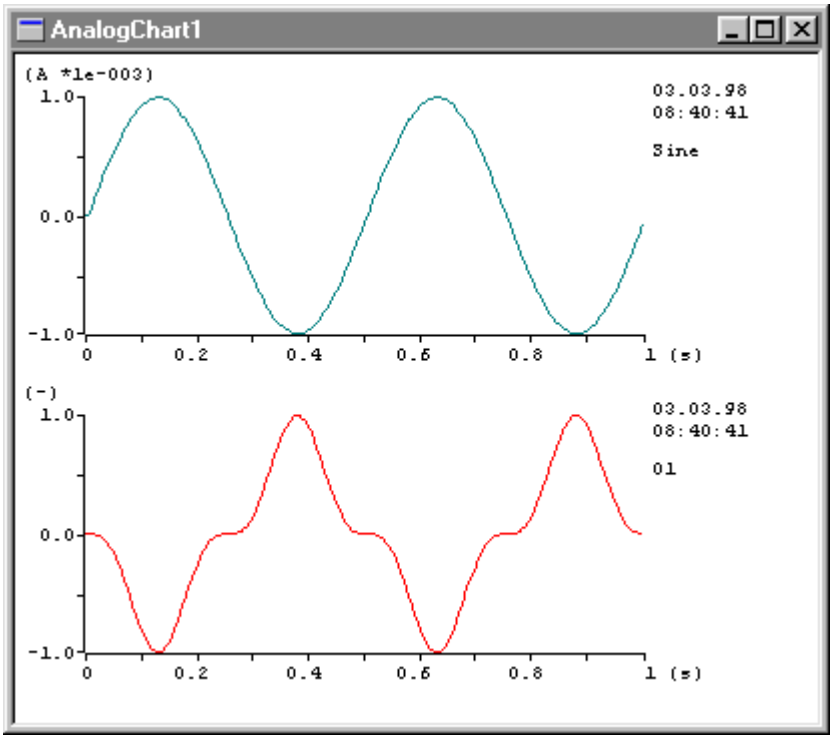
This generates a second input. The expected type is a scalar of data type integer. **Pow(base,exponent)** defines the power function. **Formula** contains a comprehensive functions library, the notation of which is similar to the syntax of the C programming language. Please refer to the online help for a complete functions reference.

The data for input **I2** are generated with the **Spin** module from the **Module > User INPUT** menu. The control element of this module contains a **spin-control**. With left-clicks on the arrows the value of the output is increased or decreased by 1. In the Spin module dialog (see pic. 3.13) the signal type and the limits of the spin counter can be set. Apart from the signal name, which is **Exponent** in the example, the initial values are not changed.



*Pic. 3.13: Dialog of the **Spin** module*

Start the program. Use **Spin** to vary the exponent of the power function, and observe the effect this has on the signal characteristics.



*Pic. 3.14: Signal characteristic in the **demo_07** program*

Please note:
*In addition to the type the complete **Type-Info** may also be entered for the output. Formula does not calculate the units and ranges[1] automatically.*

*The **demo_08** program uses the following extended **Type-Info**.*

O1[f,name = "P", min = -1, max = 1, unit = "V", scale = 0.001] = pow(I1,I2[%d])*-1;

***P** is the name for signal O1. The value range of the signal is defined from min = -1 to max = 1. A unit **V** (volt) does not make sense physically, but is suitable for demonstration purposes. The scaling factor **scale** is 0.001. It must also be noted that, if the **Type-Info** is entered, the type of the output signal also has to be entered.*

*You can define several outputs, i.e. several formulas, within a **Formula** module.*

Please note that the exponent notation (e.g. 1E-2 for 0.01) is not possible here.

The next chapter describes the modules of the **Display** group in more detail. From this group the modules **DigitalDisp** and **AnalogChart** have already been used. The chapter also explains how a user interface can be designed.

3.2 Modules of the Display Group

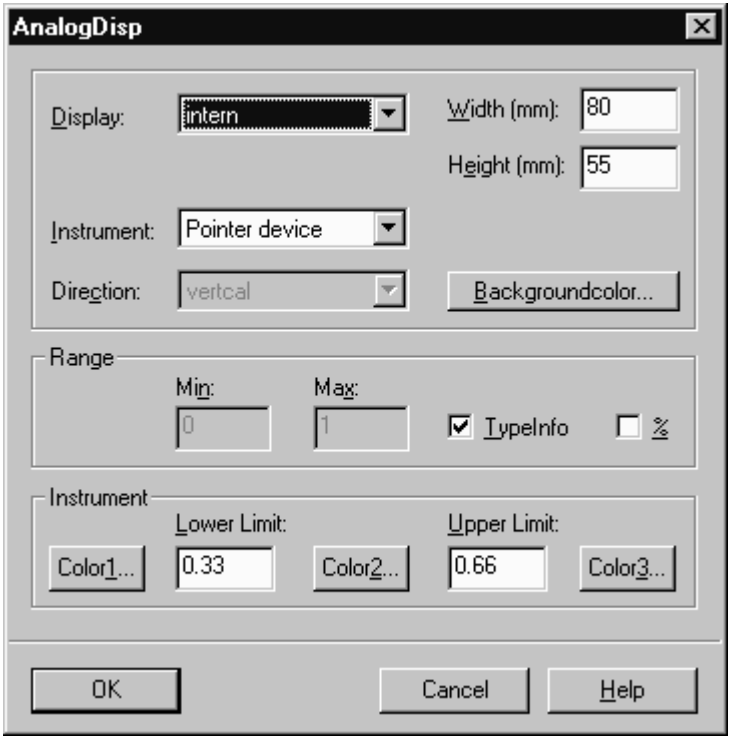
In this chapter you will learn about the visualisation possibilities of ICONNECT. You will also see how a user interface can be designed with the **DisplayManager** and **InputManager** modules.

3.2.1 The AnalogDisp Module

First another possibility of representing individual values with the **AnalogDisp** module shall be demonstrated. The module can be configured either as a pointer instrument or as a bar display. For the display range you can determine

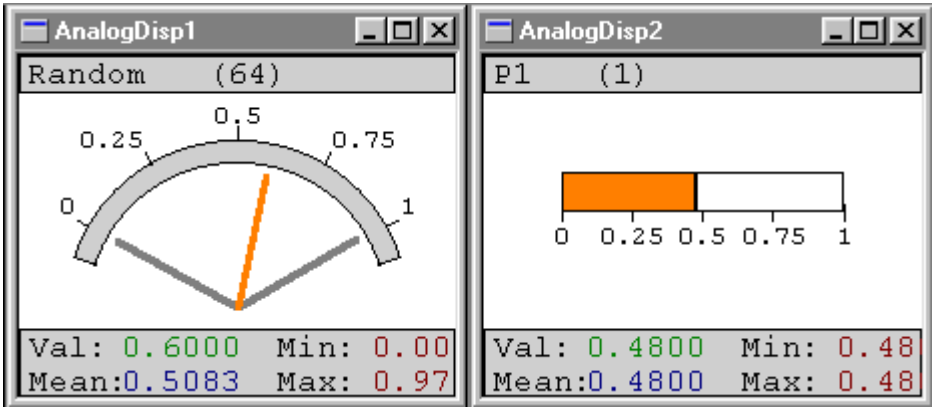
- fixed limits, or
- the limits of the **Type-Info**.

Both variants can be displayed in percent.



*Pic. 3.15: Dialog of the **AnalogDisp** module*

The **Display** parameter determines whether the module has its own display window, or whether it is integrated in the display manager. If you select **Display > extern** for a visualisation module, the module icon in the signal graph will receive an output that can be connected to the **DisplayManager** module.



*Pic. 3.16: The **AnalogDisp** module as pointer instrument with slave pointers or as bar graph*

The **AnalogDisp** module recognises the data format at the input. In its operation mode it distinguishes between a single datum or a data block with several values.

- **Val** Last value of a block
- **Mean** Mean value of a packet
- **Min** Minimum of a packet
- **Max** Maximum of a packet

If the **AnalogDisp** module functions as a pointer instrument, the **Min** and **Max** values are visualized in the form of two slave pointers.

Please note:
*The slave pointers do not refer to the complete measurement sequence. The information **Random (64)** or **P1 (1)** that is shown in pic. 3.16 displays the type of the data source and in brackets the total of the data values (= multiple of the block size).*

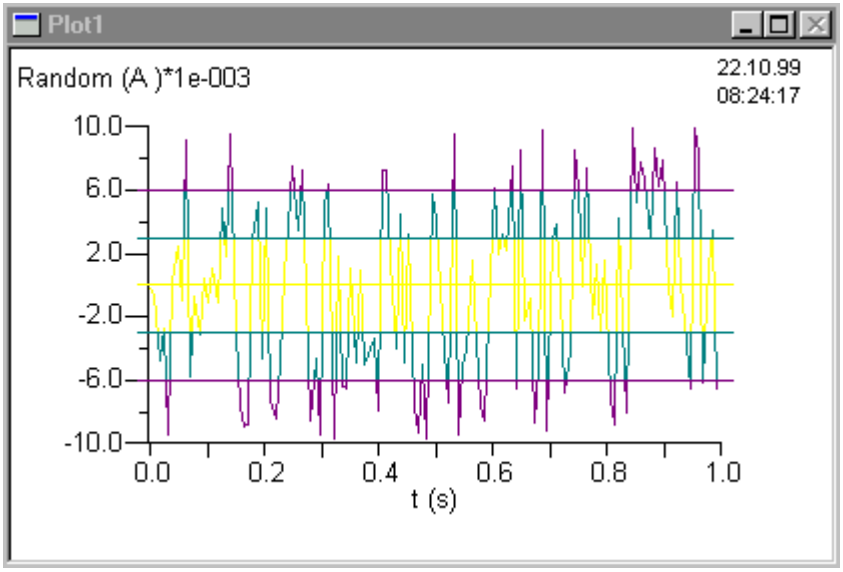
3.2.2 The Plot Module

Depending on the wiring of the X-input the **Plot** module (**Modules** > **Display** > **Array**) distinguishes between two operating methods:

X-input not connected	Y-data are plotted over time
X-input connected	Y-data are plotted over X

*Pic. 3.17: The **Plot** module*

Start the **demo_11** program. The **Plot** module visualizes the noise generated by the **Random** module. This program forms the basis for further extensions you will still add in this chapter. The noise is displayed in different colours, which allows you to distinguish between **warning** and **alarm limits**.

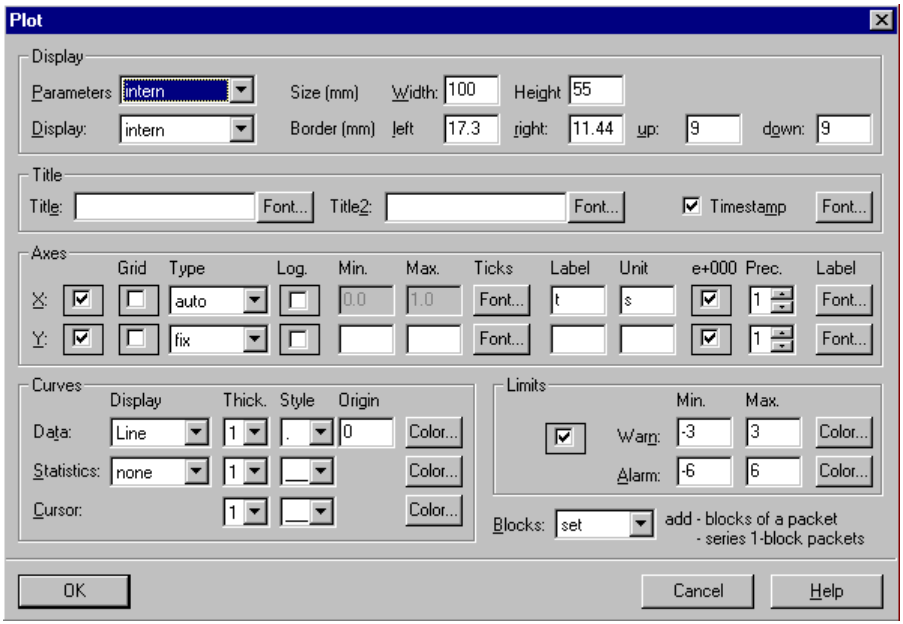


*Pic. 3.18: Display window of the **Plot** module*

Stop the **demo_11** program and open the dialog of the **Plot** module (see pic. 3.19). Type**e-Info** was chosen for Y-axis scaling. The signal name, minimum and maximum value, unit, and scaling factor correspond with the information of the data packet at the Y-input. **Auto** was chosen for the X-axis the axis scaling. From the sampling rate Plot calculates the time scale for displaying the Y-data. In Auto-Range-Mode no signal name and no unit are generated. Therefore the **Label t** and **Unit s** were entered.

Type in **White Noise** as **Title** or **Subtitle**. Use **Precision** to increase the number of decimal digits of the axis. Start the program and observe the changes.

You may also define **warning** and **alarm limits** to use the Plot module as a visualisation tool in the field of quality inspection or monitoring. If the limits are exceeded the colours will be changed. In the example the limits are set to -3, 3, -6, and 6.



*Pic. 3.19: Dialog of the **Plot** module*

As already mentioned in chapter 2.1.4 you can modify parameters during the application runtime, if the parameter source is set to **extern**. The following steps have to be performed for this purpose.

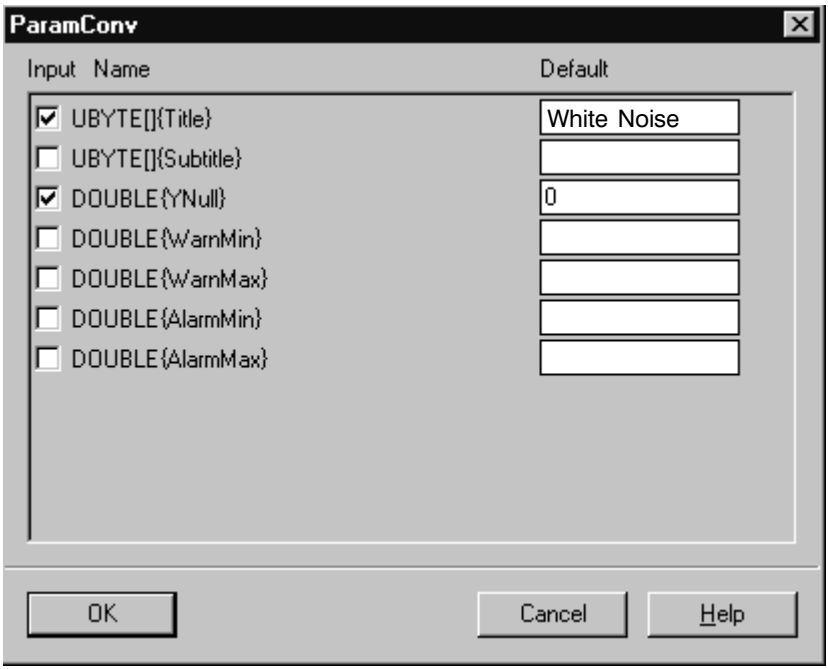
1. Select **extern** in the **Plot** module for the parameter source.

2. Insert the **ParamConv** module from the **Module > SignalProcessing > Convert** menu into the work area. Connect the output of **ParamConv** with the **EXT** input of the **Plot** module.

The **ParamConv** module operates as a parallel-serial-converter. **Plot** expects the parameters at the input in serial form.

The pop-up window of **ParamConv** displays all the signals the **Plot** module expects. For each signal you may define a fixed value or a control input.

3. Open the pop-up window of **ParamConv**, select the line **UBYTE[]{Title},** and type in **White Noise** in the **Default** field (see pic. 3.20).



*Pic. 3.20: Dialog of the **ParamConv** module*

4. Double-click-left on the line **DOUBLE{Origin}.** Type in the value **0** in **Default value**, and confirm your entry with **OK**.

With steps 3 (**Title**) and 4 (**Zero point**) you have now assigned fixed values for the **Plot** module. ICONNECT enters this in the dialog of the **ParamConv** module. In the **Control** column you will find the entry **Value**.

The warning and alarm limits should be variable during runtime. Therefore the **ParamConv** module must be equipped with control inputs.

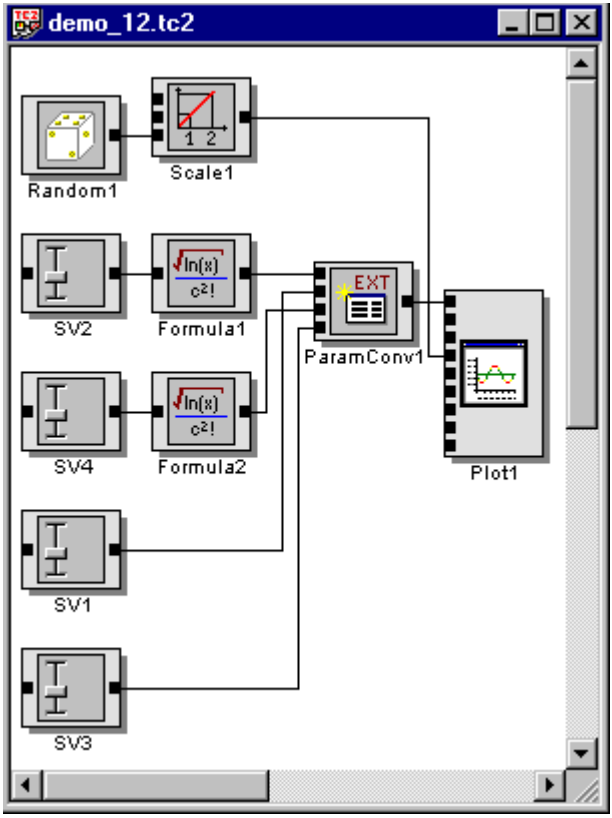
5. Open the pop-up window of **ParamConv**, select the lines **DOUBLE {WarnMin}, DOUBLE {WarnMax}, DOUBLE{AlarmMin},** and **DOUBLE{AlarmMax}.**

Two potis are connected directly to the inputs **WarnMax** and **AlarmMax**. The Formula module negates the inputs for the values (**WarnMax** and **AlarmMax**).

6. Complete the signal graph (see pic. 3.21). Enter signal type **DOUBLE** and input **Intern** for the **Poti** modules. Edit the command line for the two **Formula** modules with

01[%f]=-I1[&f];

When you start the signal graph now, you can adjust the warning and alarm limits during the application runtime with the four potis.



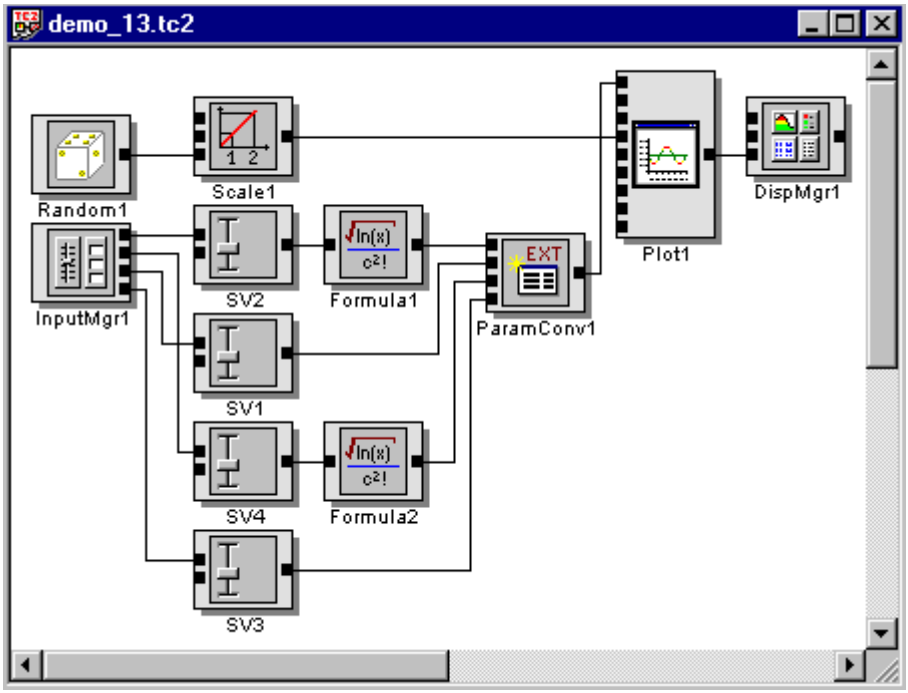
*Pic. 3.21: Signal graph of **demo_12***

Apart from the signals in the time range ICONNECT can also represent binary signals. ICONNECT provides two modules for visualising binary signals. The functions of **DigitalChart** essentially are the same as those of **AnalogChart**, they will therefore not be discussed in detail any further. **BinaryDisp** will be discussed with the **Logic** group.

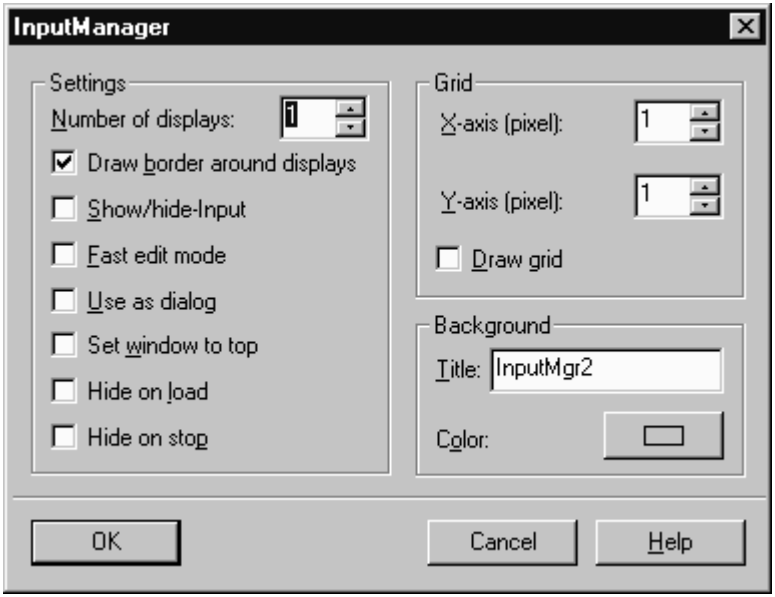
3.2.3 The InputManager and DisplayManager Modules

The **InputManager** and **DisplayManger** modules are used to group control elements or output elements, respectively. The signal graph of **demo_13** (see pic. 3.22) uses a simple user interface. As already mentioned above visualisation modules can be set in such a way that they can be connected with the DisplayManager. In an analog way modules with control elements can be connected with the InputManager.

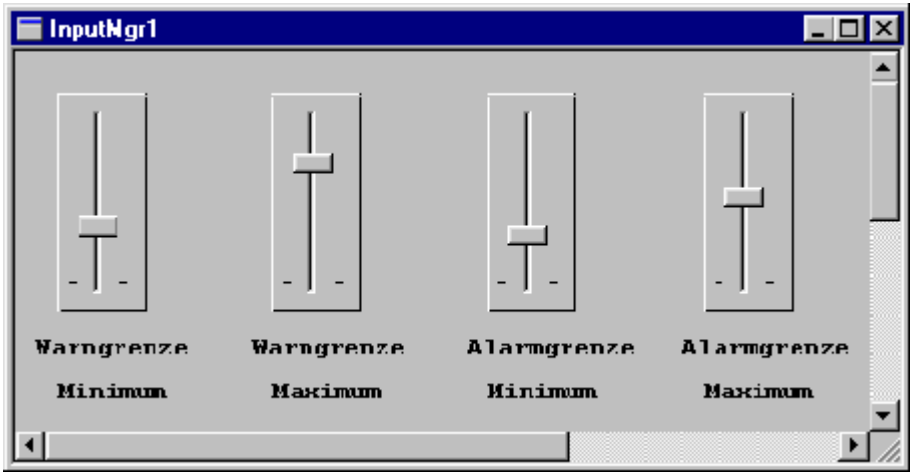
The number of outputs for the InputManager can be defined in **Number of displays** (see pop-up window of the **InputManager** in pic. 3.23). The control element is accepted in the InputManager as soon as an output of the **InputManager** module is connected with the **Poti** module (see pic. 3.24).



*Pic. 3.22: Signal graph of **demo_13***



*Pic. 3.23: Dialog of the **InputManager** module*



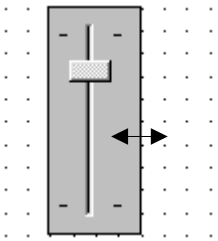
*Pic. 3.24: Control interface of the **InputManager** module*

The size and position of a control element can be adjusted with the mouse. This is done as follows:

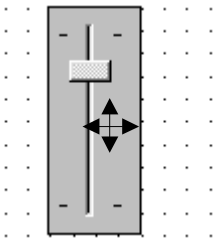
1. Double-click-left in the window of the InputManager

The white backgrounded window indicates that you are in the edit mode. A suitable grid (see **Draw grid** in the **InputManager** dialog) facilitates the customisation of the input window.

2. Place the mouse pointer on the edge of the control element (see pic. 3.25a) and vary the size with pressed left mouse button. If you want to change the position of a control element, place the mouse pointer over the control element (see pic. 3.25b) and move the control element to the desired position with pressed left mouse button.



Pic. 3.25a: Resize



Pic. 3.25b: Move

A single right-click opens a pop-up menu which offers the possibility of entering text, a line, or a rectangle. An already drawn object can be deleted by means of the context menu. The edit mode is cancelled with a double-click-left.

ICONNECT offers the possibility of hiding the window for the InputManager as an option. In connection with logic modules and control signals (see chapter 3.3) this allows you to define any dialog sequences for the user interface.

The principle of the DisplayManager is equivalent to the InputManager. The procedure for designing interfaces also is the same. The DisplayManager, however, has an additional input which prints the current status of the DisplayManager on a printer when it is activated by a control signal (see chapter 3.3).

The next chapter explains the logic modules and the difference between data signals and control signals will also be discussed.

3.3 Modules of the Logic Group

With the help of several examples you will

- learn to perform control task in ICONNECT
- learn about new modules of the **User INPUT** and **Display** group

3.3.1 The Count Module

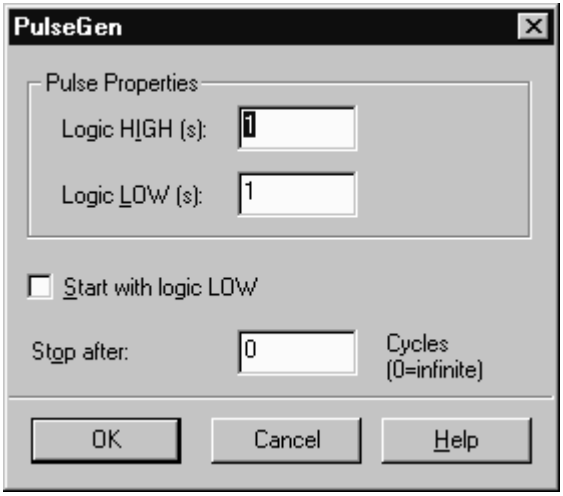
Count either counts
- blocks or
- packets (see pic. 3.28)

Input **I1** of the module is compatible with any data types.

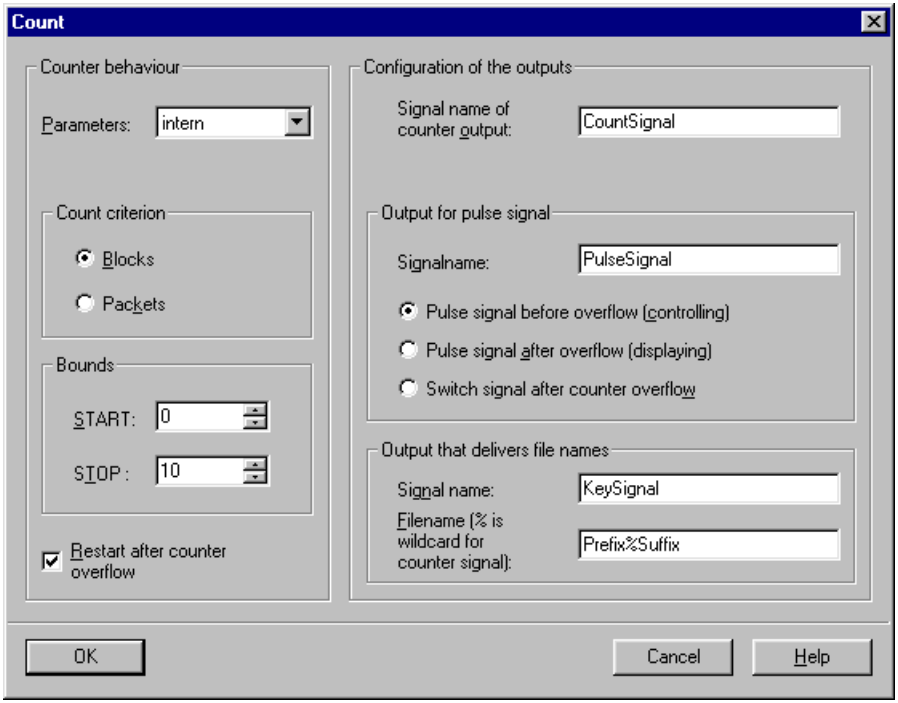
*Pic. 3.26: Control element of the **Button** module*

Start the **demo_14** program. The pulse generator in the signal graph of **demo_14** is started with a left-click on the button **B2** (see pic. 3.26) from the **Module > User INPUT** menu. **PulseGen** supplies pulses with the values **Zero** and **One**. The signals are generated with the time behaviour set in the dialog (see pic. 3.27). The parameters are chosen in such a way that **PulseGen** generates a positive edge every two seconds. The signal is routed to the count input I1 of the **Count** module. The count of the **Count** module increases with every edge from the pulse generator and is sent to the corresponding output. The **DigitalDisp** module visualizes the count.

Please note:
Switching signals always have packet status.



*Pic. 3.27: Dialog of the **PulseGen** module*



*Pic. 3.28: Dialog of the **Count** module*

In the dialog of the **Count** module (see pic. 3.28) the values 0 and 10 are entered as the counter limits. The counter will be restarted after an overflow. Apart from the output of the count the count can also generate control signals.

In the signal graph of example **demo_15** the control signals of the **Count** module are displayed with the **BinaryDisp** module. **BinaryDisp** displays the status of binary signals with LED's. As with the digital display it is also possible to specify several inputs. Furthermore you may assign names to the two signal states.

File names are used to control the file access of certain modules (see chapter 3.4).

The term control signals was used several times above. Please observe the difference between data signals and control signals:

- Control signals are one-dimensional vectors of type **SWORD[1]** and have the interpretation **BIN**. The logic modules and all the modules that communicate with control signals immediately respond to a change at an individual input.

- Data signals are multi-dimensional vectors that have been created with a certain sampling rate. The modules will only process such data, if data are present at all the corresponding inputs. Only status changes are transmitted.

3.3.2 The UniGate Module

The logic gates can be found in the **Module > Logic** menu. These gates realise logic operations (logic algebra). In all the modules of the **UniGate** group you may invert both the inputs and the outputs.

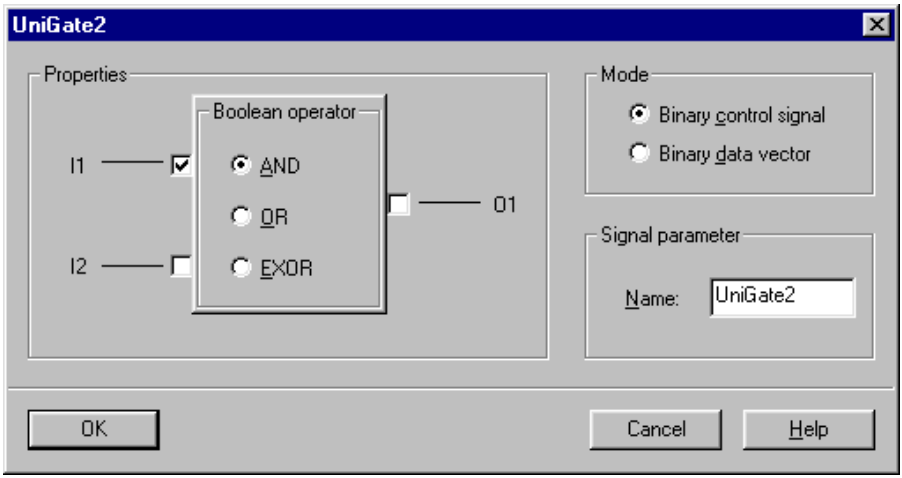
Type	Inputs	Logic operation		
		AND	OR	EXOR
UniGate2	2	✓	✓	✓
UniGate3	3	✓	✓	
UniGate4	4	✓	✓	

Tab. 3.2: Possible operations of the UniGate module group

Pic. 3.29 shows the configuration for a logic AND operation with two inputs. Input I1 is inverted in this case.

Short notation:

$$O1 = \overline{I1} \wedge I2$$

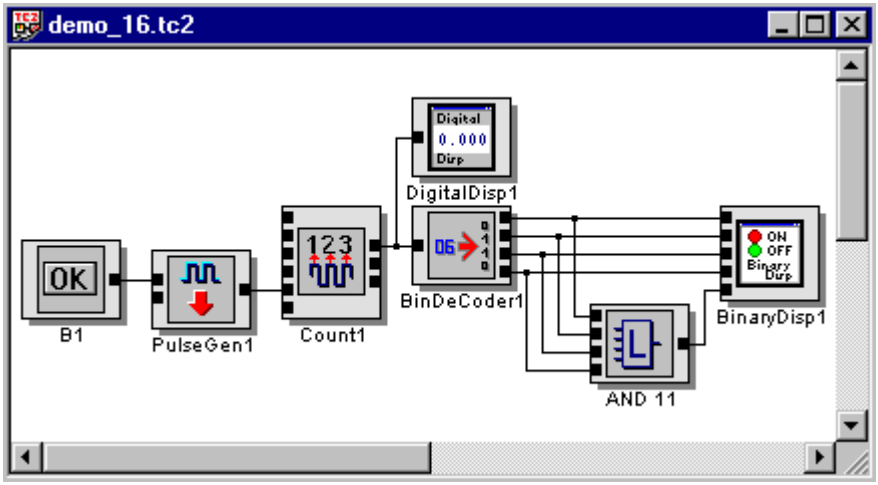


*Pic. 3.29: Dialog of the **UniGate2** module*

The example **demo_16** generates all the possible input combinations for the **UniGate4** module. The **BinDeCoder** module converts a decimal number into binary signals. The **BinDeCoder** module can be assigned the following coding types:

- Decimal → Binary
- Decimal → BCD (8421 code)
- Decimal → Grey code

Start the **demo_16** example. The BinaryDisp module displays the states of the four inputs and the result of the operation.



*Pic. 3.30: Signal graph of **demo_16***

Please note:
*The logic operations NAND, NOR, and XOR can be realised with the basic functions AND, OR, EXOR, and NOT. For detailed literature on the subject of logic algebra (Boolean algebra) please refer to the book **Halbleiter-Schaltungstechnik** by **Tietzke/Schenk**.*

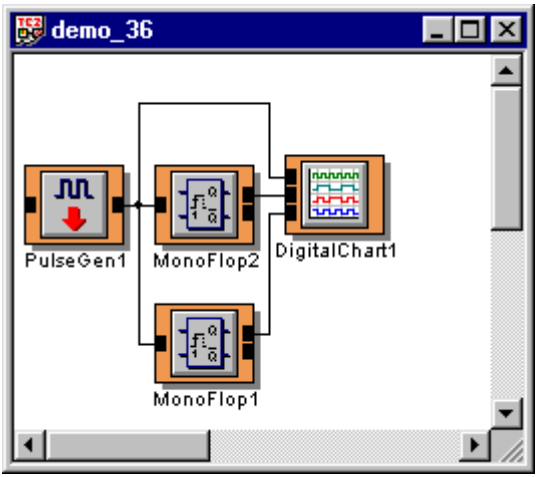
3.3.3 The FlipFlop, Mono-Flop, and T-FlipFlop Modules

ICONNECT flipflops simulate circuits used in electrical engineering. They have the ability to function as a storage element. They are able to take up certain states without time limitation. At present the following flipflop types are realised:

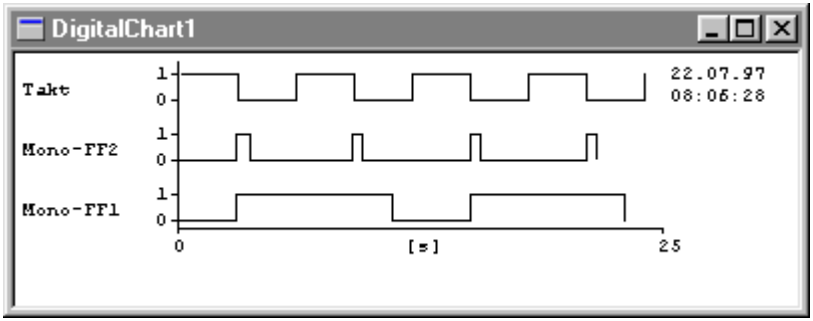
- RS-flipflop — Set-reset flipflop, basic flipflop type
- Mono-flop — Pos. or neg. edge triggering, variable hold time from 0.01 s to 10,000 s, retriggerable
- T-flipflop — Pos. or neg. edge triggering

The signal graph in pic. 3.31 shows the gating of two mono-flops. The flipflops are triggered by the negative edge of the **PulseGen1** module.

Both mono-flops synchronously take up the logic status **One**, but due to different hold times they drop one after the other (see pic. 3.32). The **DigitalChart** module visualizes the signals. Start the **demo_36** example, vary the parameters of the modules and observe the effects.



*Pic. 3.31: Signal graph of **demo_36***



*Pic. 3.32: Signal characteristic of **demo_36***

Please note:
*The input combination **Set[0]** and **Reset[0]** is not defined for the **RS-flipflop**. In this case the **RS-flipflop** supplies the output state of the last valid input combination.*

3.4 Saving and Loading Data

In this chapter you will learn

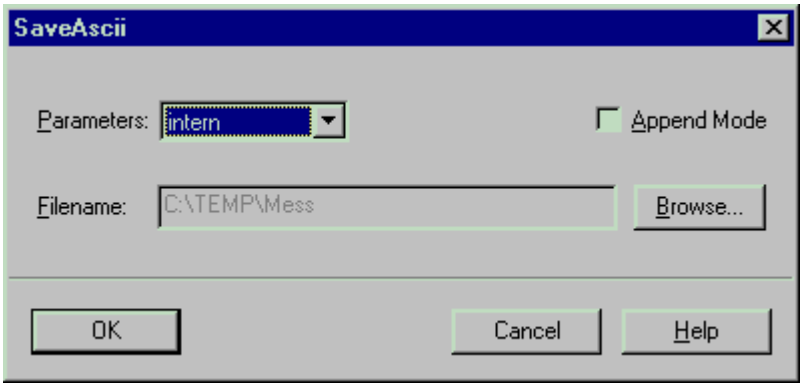
- how to save data in files and load data from files
- how to use the different file formats **SaveAscii**, **LoadAscii**, **SaveTable**, **LoadTable**

ICONNECT offers two different file formats. The **SaveAscii** and **LoadAscii** modules from the **Module > Data I/O > File** menu save and load files in the ASCII format. **SaveAscii** adds additional time information. With **LoadAscii** you can load data in accordance with the time behaviour they were saved.

The **SaveTable** and **LoadTable** modules save and load data in the form of tables. A column of the table corresponds with a module input. The format is compatible with spreadsheets in office packets, such as e.g. MS Excel.

Since the parameterisation of the two types is almost equivalent, we will only describe the process of saving and loading ASCII files here.

Load the **demo_17** example, and open the pop-up window of **SaveAscii** (see pic. 3.33). Determine the path and the file name of the file under the item **Browse...** . The file name will be determined during application runtime. An initial file name must still be defined, because when the application is started a file with the initial file name will be opened.



*Pic. 3.33: Dialog of the **SaveAscii** module*

With the **Apend Mode** item you can determine whether the data will be added at the end of the already existing file after an application restart, or whether the old file will be overwritten.

You may also assign the file name depending on the runtime through the **EXT** input (= external input). For example the file name may be generated automatically with the **Count** module. In this case the count is connected with a key word. This requires the following steps:

1. Open the dialog of the **Count** module. In the field **Output that ... > Filename** type in the path **c:\Temp\Mess%.dat** for file saving.

The path information consists of

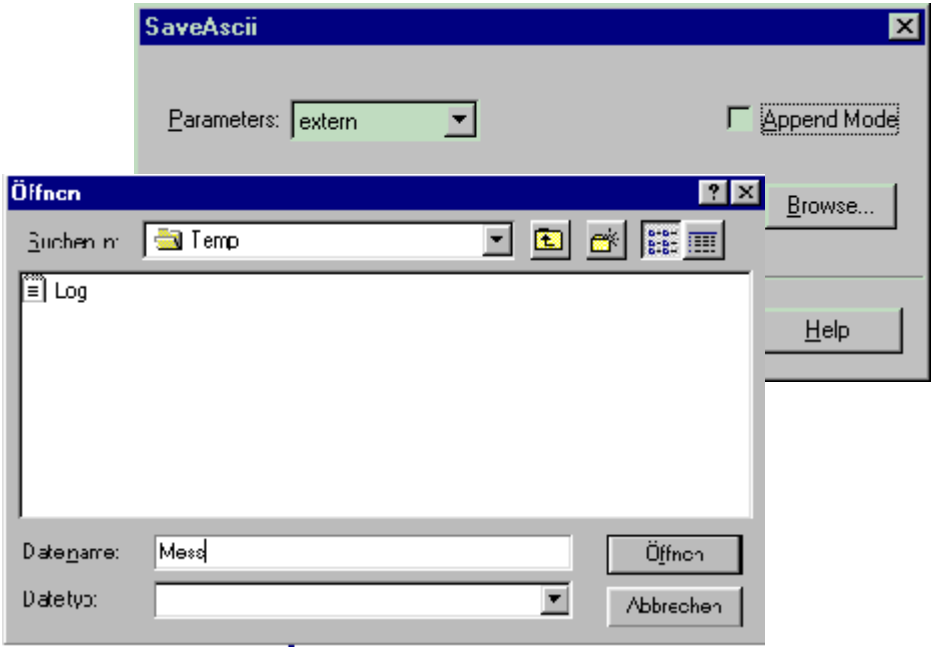
Prefix	Path and file name
%	Count
Suffix	File extension

In the example select **c:\Daten\Mess** as the prefix, and **.dat** as the suffix. During key word generation the count will be entered at the position of the percent sign. This means that a definite file name will be generated for every measurement.

2. Open the dialog of the **SaveAscii** module (see pic. 2.34). Type in the path **c:\Daten\Mess**.

Start the program and check the generated.

Please note:
ICONNECT does not generate any paths. Please select an existing path or generate a path, if necessary.



*Pic. 3.34: Dialogs of the **SaveAscii** module*

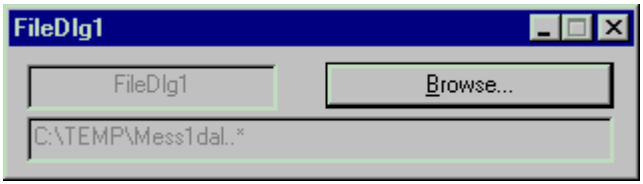
The **FileDlg** module is another possibility for generating a file name during application runtime. This module calls the Windows file dialog, it is primarily used for reading files.

In the **demo_17** example you created files. ICONNECT offers the possibility of viewing saved measurement values at a later time. These files shall now be loaded. Please perform the following steps:

1. Load the **demo_19** example.

2. In the signal graph open the dialogs of the **FileName** and **LoadAscii1** modules. Select the path you have assigned to the measurement data of **demo_17**.

3. Start the program. Press the **Enter** button (see pic. 3.35) in the control element of the **FileDg1** module. This starts the Windows file dialog for selecting the desired measurement file.



*Pic. 3.35: Control element of the **FileDlg** module.*

Please note:
*The appendix (see chapter 9.2) contains more information on the subject of **saving and loading data**.*

3.5 Modules of the Signal Processing Group

In this chapter you will learn about the possibilities

· for limit value monitoring
· for cutting data
· for generating trigger events
· for filtering and smoothing data

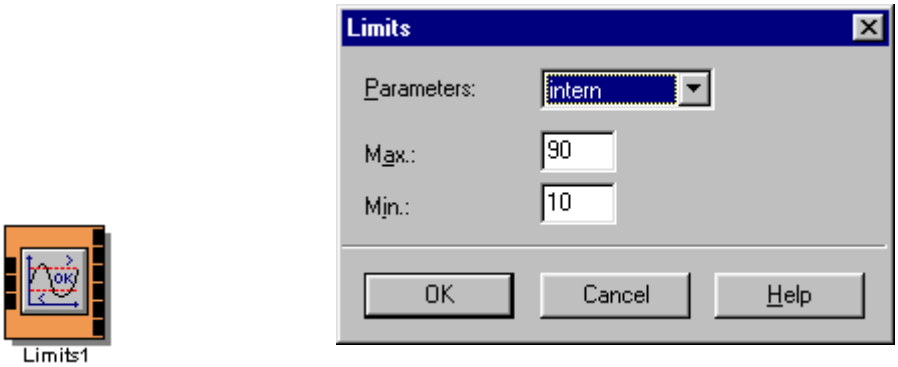
At the end of the chapter some mathematically more complex modules like FFT (Fourier transformation) and CFFT (complex Fourier transformation) will be introduced.

The Limits Module

The Limits module can be used for solving different tasks.

Limit value monitoring

The **demo_20** example checks whether a signal exceeds certain limit values. In the **Limits** dialog (see pic. 3.36) two limit values can be defined. For limit value monitoring the two outputs **S>** and **S<** are connected. These outputs supply a binary signal of type **SWORD[1],** if the range given in the dialog is exceeded in upward **S>** or downward **S<** direction.

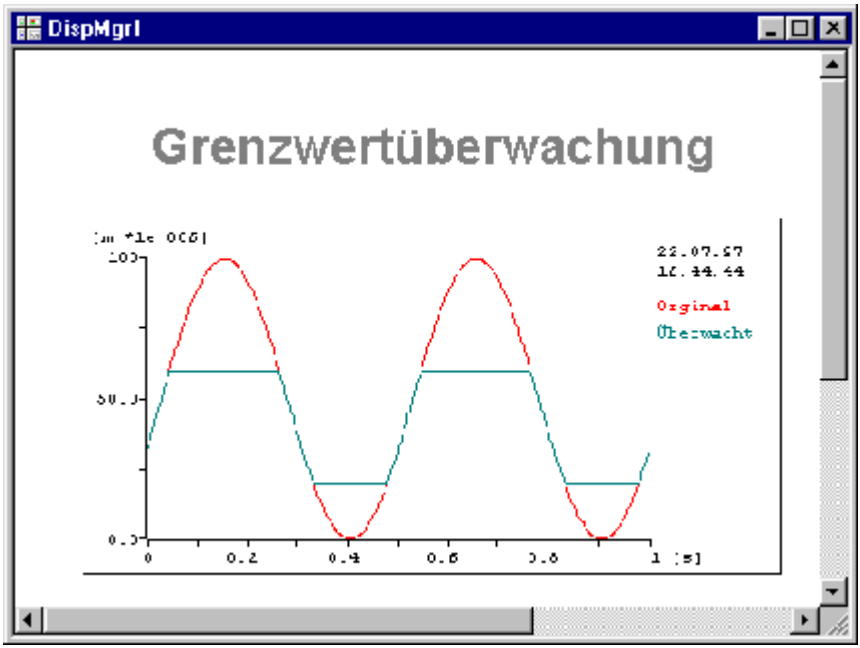


*Pic. 3.36: Icon and dialog of the **Limits** module*

The modules **FuncGen** and **Scale** generates a displacement signal with a measuring range of 100 μm. In the limit value module a range from 10 μm to 90 μm is defined. If these limit values are exceeded in upward or downward direction, this will be displayed by two **BinaryDisp** modules. The complete visualisation is performed in the display manager.

Signal limitation

The **Limits** module can be used for limiting signals. When the permissible range is exceeded **Limits** will keep the limit value until the original signal lies within the limit values again (see pic. 3.37). In this case the **LIM** output (see **demo_21** example) must be connected.



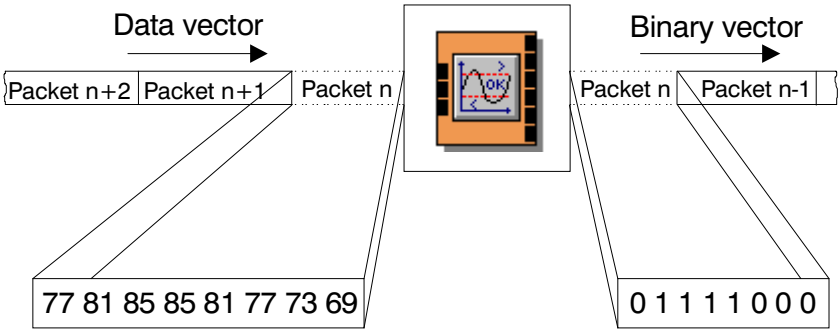
*Pic. 3.37: Signal limitation with **Limits***

Cutting of signal parts

In combination with **Limits** the **Edges** module can cut signal parts. For this purpose the signals must be present in packet form.

Limits generates binary vectors at the outputs **>**, **OK**, and **<**. These vectors characterise the correlation between the limit values and the input data. In the **demo_22** example the output **>** of **Limits** is connected with the input **BIN** of **Edges**. **Edges** then manipulates the original signal at input **I1** with the binary vector. If the datum at the index position x of the input vector lies within the permissible range, the value 0 is written into the output vector at this place. If the value lies above the specified limit value, a 1 will be entered.

For the part of the data to be cut the range must be correspondingly defined in the dialog of **Limits**. The generated binary signal is passed to the **Edges** module.



*Pic. 3.38: Generation of a binary vector by the **Limits** module*

Demo_22 implements an edge detection. The program simulates input data in the range from 0 μm to 100 μm. All the data of the original signal that lie between the first entry and the last exit of the original signal into and from the range above 80 μm will be cut.

For some tasks it is necessary to remove the edges of a cut out range in order to gate signal irregularities. A start and stop limit in percent can be defined in the **Edges** dialog. In the **demo_22** example the first and the last 5% of the signal are gated.

Another possibility is to cut out only the part of the signal that lies above the limit value (see **demo_23** example). For this purpose select the **Block of 1-signals** in the **Edges** module and enter the second block. This parameter set cuts out the peak of the second sine wave.

Please note:
As an alternative you can cut out signal parts by replacing ***Edges*** *with* ***Packet****, and* ***Limits*** *with* ***Trigger****.*

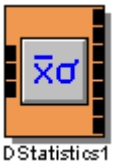
Trigger*, however, does not expect data in packet form. With this you can derive certain events from data that are recorded in continuous mode.*

3.6 Modules of the Statistics Group

3.6.1 The DStatistics Module

DStatistics calculates descriptive statistical values. At present these are:

- Number, minimum, maximum
- Mean value, standard deviation
- Skewness, kurtosis



*Pic. 3.39: The **DStatistics** module*

The inputs are:

- **I1** Data input
- **Cal** Calculate
- **Res** Reset

The statistical output values are calculated and output as soon as
- a **One** signal is present at the **Cal** input, or
- a **High** level is present at the **Res** input, or
- a packet end is achieved.

The statistics results are cumulated until
- the **Res** input is reset, or
- the measurement is stopped.

Please note:
High at the Reset input does not lead to a reset of a display that is connected with DStatistics

In the **demo_24** example the **Count** module generates a pulse signal at every second block effecting the calculation of the descriptive statistical values. The **Reset** button is used to reset the buffer of the **DStatistics** module.

3.6.2 The Hist Module

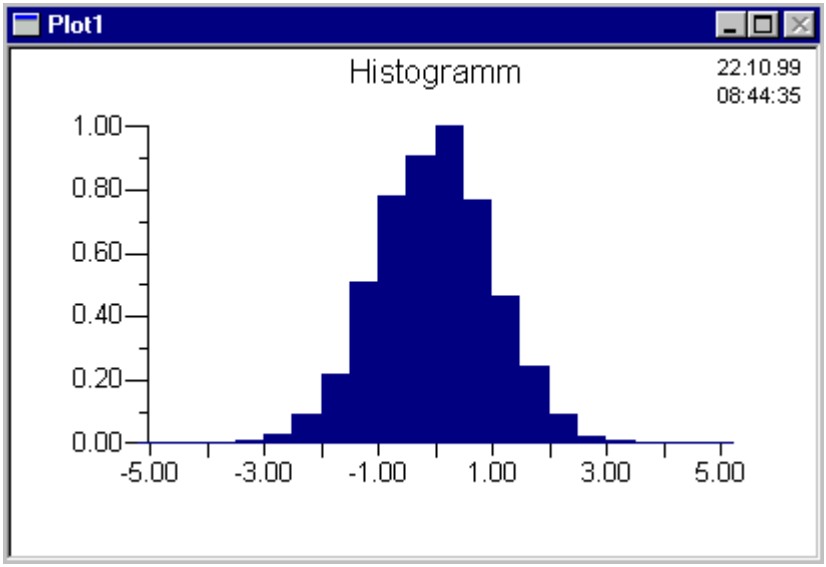
Frequency distributions are used for monitoring the quality of, for example, a production process. The **Hist** module evaluates the abundances of the individual events.

*Pic. 3.40: The **Hist** module*

Hist features the following parameters:

- Range — Value range to be displayed in the histogram
- Classes — Number of possible discrete events on the abscissa
- Scaling — Scaling of the absolute frequency on the ordinate

The **demo_25** example evaluates the series of numbers of a random number generator. The **Plot** module visualizes the results of **Hist**.



Pic. 3.41: Histogram of a random number generator

3.6.3 The Sort Module

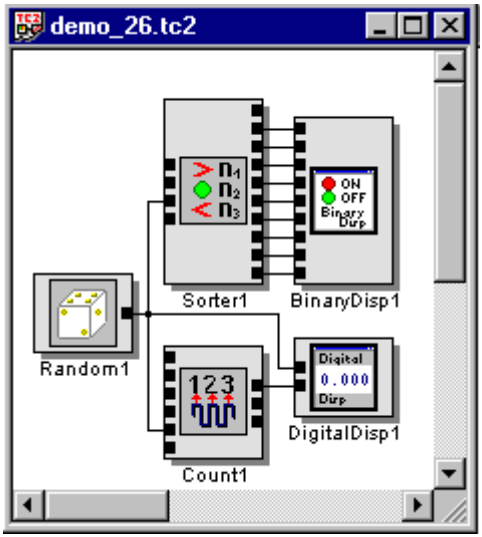
The **Sort** module is used to solve control tasks. In its dialog you can define

minimum and maximum for the value range
the number of sorting classes
the class limits
the type of the output signal

Based on the value range and on the number of sorting classes **Sort** calculates the class limits. If you want to select the class limits manually, double-click on the class limit to be changed and edit this.

As the output quantity you either select a vector of type **DOUBLE[]**, which determines the number of values belonging to this range for every sorting class, or a binary control signal **SWORD[1]**, which can, for example, control a sorting station.

In the **demo_26** example (see pic. 3.42) random numbers are supplied to the Sort module for sorting. The sorting module generates a **One** signal at the output for every value corresponding to the class.



Pic. 3.42: Sorting of random numbers

3.7 Modules of the FlowControl Group

In this chapter you will learn some functions for controlling the program flow in ICONNECT.

- Software loops — Implementation of adaptive algorithms
- Multiplexer — Assignment of data to different partial graphs

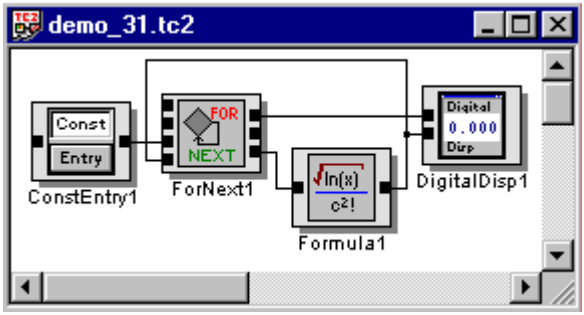
With these functions you can split up an application into different processing phases.

3.7.1 The ForNext Module

ForNext generates a loop with defined limits. The **demo_31** example (see pic. 3.43) calculates the value

E = 100 * 4 + starting value

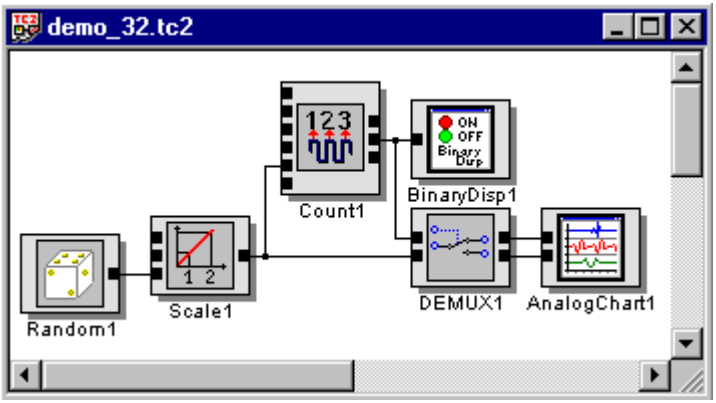
in a loop. In the dialog of the **ForNext** module 0 is defined as the lower loop limit, and 99 as the upper loop limit. As data type a vector of type **DOUBLE[]** is processed. In the formula interpreter (**Formula** module) the calculation 01 = I1 + 4 is performed, i.e. the multiplication is reduced to an addition. With every loop run the value 4 is added to the sum formed so far.



*Pic. 3.43: Signal graph of **demo_31***

3.7.2 The DEMUX Module

DEMUX is able to direct data into different partial graphs. In the **demo_32** example (see pic. 3.44) nine out of ten data packets are directed to the first display, and one is directed to the second display. The counter **Count1** generates a 1-signal at its pulse output before counter overflow. This signal is used to switch the demultiplexer.



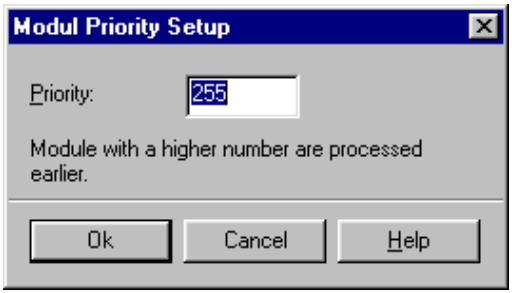
*Pic. 3.44: Signal graph of **demo_32***

It must be ensured that the control data (switching of the demultiplexer) and the signal data are correctly synchronised.

The **DEMUX** module is ready to perform calculations (see also chapter 6.2), if data are present at input **I1**. In order to guarantee that the demultiplexer switches at the right moment, the pulse signal must not arrive at the module after the data from the random number generator. Otherwise it might be possible that **DEMUX** still transports these data before the pulse signal arrives. **DEMUX** is ready to perform calculations as soon as there are data from the random number generator. The counter also is ready as soon as Random provides the data.

The module with the higher priority is executed first. The priority is encoded with 16 bit and may have values from 0 to 65535. The default value is 255. In the example the **Count** module has the priority 256.

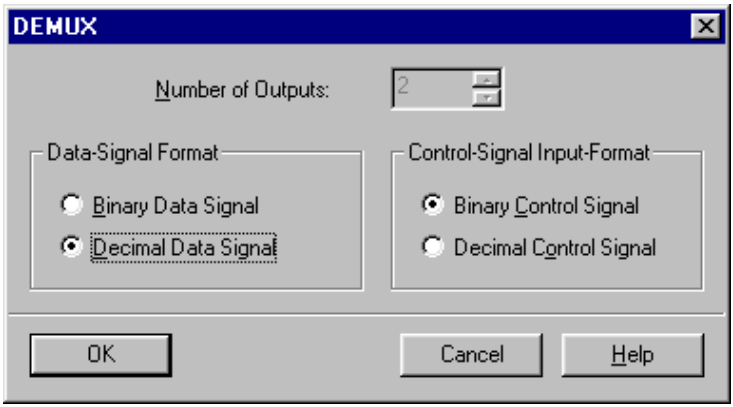
A right-click on the **Count** module icon opens a pop-up menu, where the **Priority...** setting can be activated with a left-click. Type in the priority of the module (see pic. 3.45).



Pic. 3.45: Dialog for priority setting

This determines that the counter is executed before the demultiplexer. **DEMUX** will only calculate, if both the data from **Random** and the pulse signal from the counter are present.

Please note:
This example is the principle design for a measurement arrangement that performs a calibration measurement once within ten measurements.



*Pic. 3.46: Dialog of the **DEMUX** module*

The demultiplexer processes the data types **SWORD[]** and **DOUBLE[]**. The multiplexer may be controlled both by binary and by decimal signals. These data types can also be transferred by the demultiplexer. The data type can be chosen in the dialog (see pic. 3.46).

With a decimal control signal the number of data outputs can be varied. It is then possible to perform demultiplexing with any number of outputs.

In the **demo_33** example the **MUX** multiplexer is used. In this example two data sources are synchronised together on one display. This is the principle design of a signal graph that combines two alternative measuring stations in one processing unit.

In this chapter the most important principles of ICONNECT were introduced. The chapter also provided an overview of the module library. The reader should now be able to generate small applications with ICONNECT himself. The next chapter will summarise information about the TypeInfo and about communication between the modules.

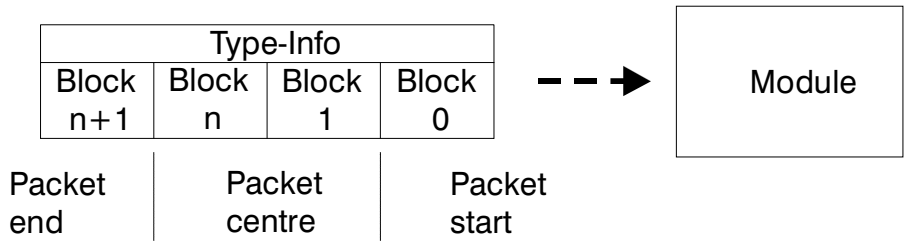
4. Communication and Type Information

Communication in ICONNECT is performed in blocks. The smallest unit is a single data value generated by a module managing a data source. Such a data source may be a HW-interface such as for example an I/O board or a file I/O module or a similar module.

The calling of a module causes certain management efforts. Calling by a single data value consumes unnecessary computer resources. In order to keep the number of calls as small as possible the data are gathered to form blocks. The size of such a block must always be set depending on the application.

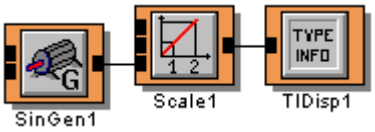
The packet mechanism was introduced for synchronizing data blocks in the graph. A packet is a number of blocks that constitute a complete unit. For example this may be all blocks belonging to an individual part.

If two packets meet on a communication channel, there will be more data than the computer can process. ICONNECT will generate an error message and stop the operation. With this “real time condition” the data are synchronised in the graph, because packets can not overtake each other. In case of modules that only process packets as a whole the individual blocks (block 0 to block n+1, pic. 4.1) are grouped by means of additional information that is contained in the communication channel.



Pic. 4.1: Information and data blocks in ICONNECT

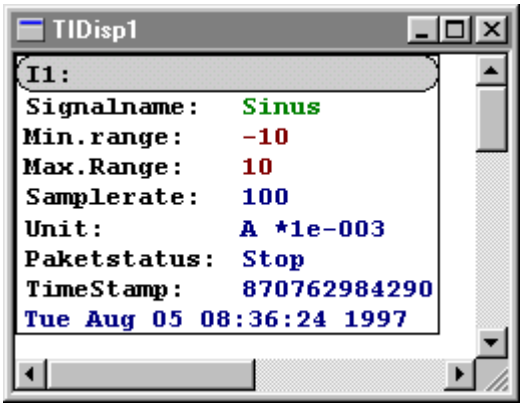
In the channel a type information (= **Type-Info**) is added to each packet, with a new type information being generated for each packet. In the **demo_34** example (see pic. 4.2) the TypeInfo display (**Module > Display > Debug > TIDisp**) is connected to the function generator. This display can be used for testing signal graphs. **TIDisp** displays the TypeInfo and the packet status of the communication channel (see pic. 4.3)



*Pic. 4.2: Signal graph of **demo_34***

TIDisp contains the following information:

- Signal name — Name of the signal
- Min. range — Lower limit of the measuring range
- Max. range — Upper limit of the measuring range
- Sampling rate — The sampling rate data are generated with
- Unit — Physical unit
- Packet status — Start, centre, stop, complete
- TimeStamp — Time stamp of signal generation



*Pic. 4.3: Type-Info of **demo_34***

In the dialog of the function generator the number of blocks in the packet can be varied. The changes of packets status during communication are visible in TIDisp module. All other information are only modified once per packet, irrespective of the number of blocks per packet.

ICONNECT modifies the units in accordance to the algorithms (see also chapter 3.1). ICONNECT uses all the SI units according to table 4.1.

Quantity	Unit name	Unit sign
Length	metre	m
Mass	kilogram	kg
Time	Second	s
Elec. current	Ampere	A
Thermodynamic temperature	Kelvin	K
Amount of substance	mol	mol
Luminous intensity	Candela	cd

Table 4.1: International units

ICONNECT derives the following units as per table 4.2.

Quantity	Unit sign	Derivation
Activity	Bq	s ^ -1
Illuminance	lx	lm m ^ -2
Pressure	Pa	N m ^ -2
Electr. Capacity	F	C V ^ -1
Electr. conductance	S	A V ^ -1
Electr. voltage	V	W A ^ -1
Electr. resistance	Ω	V A ^ -1
Energy	J	N m
Energy dose	Gy	J kg ^ -1
Frequency	Hz	s ^ -1
Inductivity	H	V s A ^ -1
Force	N	m kg s ^ -2
Charge	C	A s
Power	W	J s ^ -1
Luminous flux	lm	cd steradian
Magn. flux	Wb	V s
Magn. flux density	T	Wb m ^ -2

Table 4.2: Derived units in ICONNECT

Apart from the packet status and the type information the data type (e.g. DOUBLE[], SWORD, ...) and its declarator (e.g. TIME_DOMAIN, BIN, ...) are also important for the communication of two modules. The following data types are used for module communication:

- DOUBLE 8 byte floating-point number
- FLOAT 4 byte floating-point number
- SWORD 4 byte integer (with sign)
- SDWORD 4 byte integer (with sign)
- UBYTE 1 byte integer, Ascii-encoded
- UWORD 4 byte integer
- UDWORD 4 byte integer
- TYPEINFO Type Info

These types can be used to create arrays (= vectors) of definite length (e.g. **DOUBLE[3]**) or of indefinite length (**DOUBLE[]**). Arrays with different definite lengths are always incompatible. A vector of definite length is compatible with a vector of indefinite length. During runtime it will be checked in the modules whether the data can be combined with each other. For this purpose the length of the data block is contained as the first value in each data block (**Type-Info**, pic. 4.1). This way the module determines how many data are present at the input for processing.

In the **demo_35** example a data vector is explicitly represented with the **Spy** module. This module is compatible with all the data outputs listed above and can therefore be used for testing a signal graph.

Please note:
The appendix (see chapter 9.1) contains more information about data types.

5. User Administration

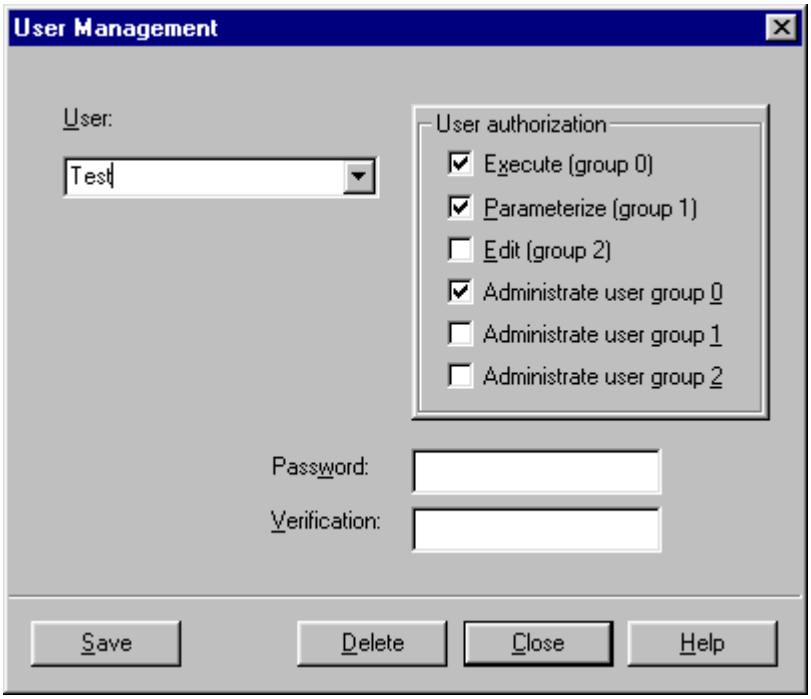
ICONNECT includes a user administration function with individual rights and passwords. There are action rights and administration rights

- Actions — Execution of a signal graph, parameterisation (setting of parameters in the module dialogs), and editing (generation of signal graphs).
- Administration — Administration of users

The execution right allows the user to start, stop, and pause signal graphs. With the parameterisation right the user can perform settings by means of a right-click with the mouse on a module, a connection, or a window background (identical with measurement - properties). The editing right allows the user to add, delete, and move modules in the signal graph. It also allows all the other editing functions from the **Edit** menu.

In the user administration the user will only see the entries that he is allowed to change in accordance with his administration rights. For example a user with three administration rights will see all the entries. A user without any administration right will not be able to open the user administration function. The user is only allowed to see and change the rights he has himself. A user without editing right can neither see nor edit the user rights. If a user has an administration right for a group, but he is not in that group (he may, for example, execute administration functions on group 2, but may not edit this group himself), the action right is also deactivated. He may neither see nor change this right in an entry.

The dialog for user administration you get by the menu **Extra > User Management**. The selected entry is displayed on the left side, and the corresponding rights on the right side. If one or several rights are deactivated (with grey background), the respective user does not have the corresponding administration or action rights. Every user must be assigned a password (at least one character), which has to be confirmed in the second line.



Pic. 5.1: User management dialog

To create a new user type in the name in the **User** field and set his rights and his password. When you have entered all data, confirm with a left-click on the **Save** button.

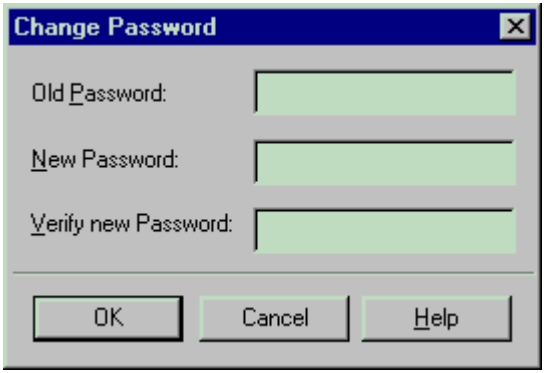
Please note:
*If the **User administration** dialog is exited without saving, the user data just entered will be lost.*

If you want to change the settings for an existing user, select this user in the **User** field. The rights can then be redefined. The new settings must be saved, again requiring password entry.

A user can be deleted by choosing him in the selection box and then by activating the **Delete** command. There will be no warning when this is done. The user will be definitely deleted.

A left-click on the **Cancel** button closes the dialog.

Every user has the right to change his password. For this purpose the old password then the new password has to be entered. An empty password may also be used, simply by entering the old password and confirming it with OK. The dialog can be found in the **Extra > Change Password** menu.



Pic. 5.2: Change password dialog

6. Tips & Tricks

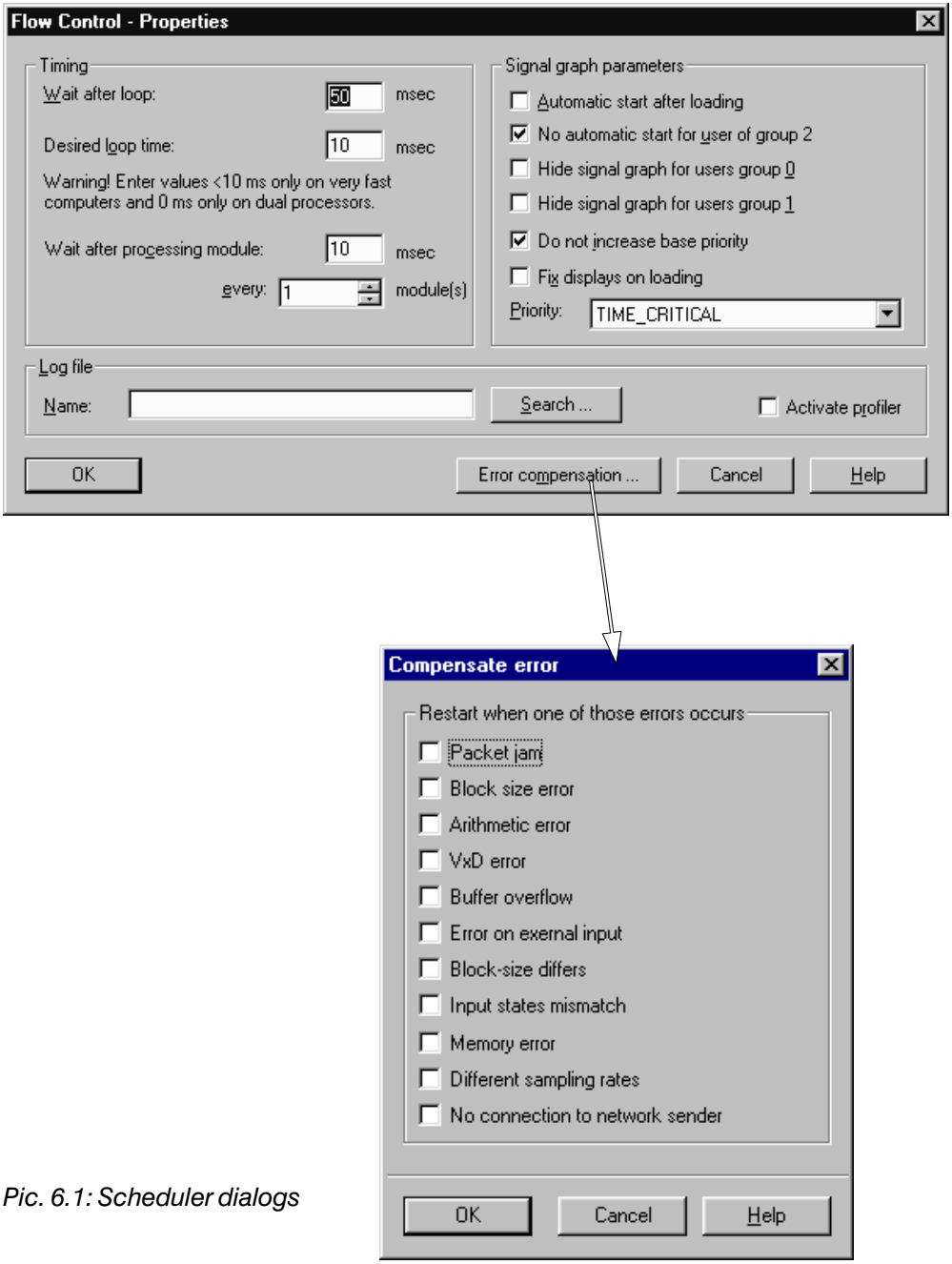
6.1 Scheduler

In ICONNECT several signal graphs can be opened and started at the same time. In this parallel type of execution several signal graphs share the processor time (pre-emptive multithreading). If CPU reconfiguration is low, all the graphs seem to run simultaneously and without influencing each other. If CPU reconfiguration increases, however, it makes sense not to distribute the calculating time uniformly, but to give preference to time-critical sequences. This can be achieved by assigning a higher priority to the critical thread.

For this purpose open the parameter dialog of the scheduler (**Measurement > Properties**). The default setting for the priority is **high** (see pic. 6.1). This parameter can be set to **low**, if required. A signal graph will only be processed, if free calculating time is available.

The low priority stage will still have preference compared to other applications. If you want to assign CPU time to other Windows applications, this must be specified in the time behaviour. For this purpose edit the **Hold-on-time after loop** field (default: 50 ms). Calculating time will be assigned prior to the cyclic processing of the ICONNECT source modules by the scheduler. The minimum set hold-on-time will be assigned to competing applications. If the nominal runtime is higher than the set hold-on-time, the computer will wait until this time is achieved. You can set a constant loop runtime that does not depend on the module calculating times.

Please note:
Since Windows is not a real-time operating system, the correct observance of time specifications can not be guaranteed! All the time settings are subject to relatively large fluctuations, especially in case of higher computer or memory reconfiguration. You should therefore never work at your computer's performance limits. Deactivate the virtual memory in the system control panel in order to avoid swapping of virtual memory.



Pic. 6.1: Scheduler dialogs

If you want to execute a signal graph automatically when ICONNECT is started, select **Autostart after login**. In the **Execution (group 0)** authorisation level your can hide the signal graph. By doing so you generate a signal graph for a runtime-version of an ICONNECT program.

The file that can be set under **Log File** records error messages (if **Restart when one of those errors occurs** is activated, see pic. 6.1) and module runtimes in Debug-mode:

```
Signalgraph FFT:***  Init         ***
Execution times for signal graph FFT
Modul: Random1     Count:15         Avg:10       Max:60
Modul: FFT1        Count:15         Avg:26       Max:60
Modul: Plot1       Count:15         Avg:0        Max:0

Signalgraph FFT:***  Done         ***
Signalgraph FFT:***  No Restart   ***
```

All the time data are given in ms (with a time resolution of the measurement of 10 ms). Apart from the average calculating times of the modules the number of calls (count) and the maximum times are recorded.

6.2 Interpret Programming

The **Interpret** module is a simple interpreter and allows the programming of individual tasks and functions. An editor for entering the program sequence appears when the dialog is started. The predefined functions are described in the online help.

An Interpret program is structured as follows:

1. Declaration of inputs/outputs and variables	
2. Initialisation phase (init)	Execution of initialisation jobs, such as for example memory allocation, or loading of drivers
3. Execution phase (execute)	Execution of the algorithm
4. Finishing phase (done)	Signal graph has been stopped, finishing jobs such as for example deallocation of memory and hardware resources

6.2.1 Input/Output Declaration

The key words **input** and **output** are used for defining module input ports and module output ports. A port declaration is structured as follows:

Port direction [trigger attribute] Port name (validation information);

Port direction:	input | output
Trigger attribute:	trigger
Port name:	String that starts with a letter and consists of letters, numbers, and "_".
Validation information:	"Data type", "Declarator" [, "Data type", "Declarator"]
Data type:	SWORD | SWORD[1] | SWORD[] | DOUBLE | DOUBLE[1] | DOUBLE[] | UBYTE[] | TYPEINFO
Designator:	BIN | TIME_DOMAIN | TypeInfo | any String

6.2.2 Declarating Variables

Variables in the Interpreter module have global validity. Initialisation during declaration is not possible.

Example:
incorrect: int i=0; correct: int i; i=0;

Variables are declared individually, comma lists are not allowed.

Example:
incorrect: int i, j, k; correct: int i; int j; int k;

Variable type Variable name ['['Array size']'];

Variable type:	char | int | double
Variable name:	String that starts with a letter and consists of letters, numbers, and "_".
Array size:	Positive integer > 0

6.2.3 Calling up Predefined Functions

All the variables of the parameter list of functions are transferred as variable references (call-by-reference). This means that no pointers are required in the Interpreter module.

Example:

```
int Len; int Address; int Status; int erg;
char recv[41];
erg = enter( recv, 40, Len, Address, Status);
```

Len and Status are return values (reference variables).

6.2.4 Program Sections

The **initialisation phase** (init) is only executed once prior to the first **execution phase** (i.e. after Measurement | Start). It is used for initialising variables or hardware components. During the **initialisation phase** access to input data is not yet possible.
The **execution phase** (execute) is performed with every program run.
The **finishing phase** (done) is processed only once - after the last execution phase
(i.e. after Measurement | Stop).

Conventions:

- All the sections are optional
- An empty program will be accepted
- At least a variable or a port must be declared before the first section
- Declared sections must not be empty
- Parameter transfers to functions and return values are not possible (use global variables!)

Example 1 (structure of an Interpret program)

```
input trigger i1 ( "TYPEINFO", "TypeInfo", "DOUBLE[]",
"TIME_DOMAIN");
output o1 ( "TYPEINFO", "TypeInfo", "DOUBLE[]",
"TIME_DOMAIN");
int i;
init
{
;
}
execute
{
;
}
done
{
;
}
```

6.2.5 Read/Write Operations on Streams

Input data can only be accessed within the execute section. Access to the input variable i1 is performed in the same way as the access to (array) variables. Access to individual elements of o is not permitted in order to ensure that the output vector variable o is completely filled. Therefore only the << operator is permitted for output:

Example 2 (input/output):

```
input trigger i1 ( "TYPEINFO", "TypeInfo", "DOUBLE[]",
"TIME_DOMAIN");
output o1 ( "TYPEINFO", "TypeInfo", "DOUBLE[]",
"TIME_DOMAIN");
int s;
int i;
execute
 {
 // o1 << i1; is permissible, access to array elements
 // is not possible
 s = size( i1);
 for ( i=0; i<s; ++i) // ++i is faster then i++
 //(i = i + 1)
  {
  o1 << i1[i] * 3.0; // output
  }
 }
```

6.2.6 The Type-Info Structure

In example 2 only data without type information are treated and output. Since the value ranges of the signal that are necessary for scaling are missing, display e.g. in the **Plot** module is not possible. Example 3 demonstrates the necessary steps for **Type-Info** treatment:

Example 3 (copying the **Type-Info**):

```
input  trigger
i1("TYPEINFO","TypeInfo","DOUBLE[]","TIME_DOMAIN");
output
o1("TYPEINFO","TypeInfo","DOUBLE[]","TIME_DOMAIN");
char name[20]; char unit[20];
int erg; int status;
double d; double timestamp; double samplerate; double
rangemin;
double rangemax; double scale;

typeinfo_kopieren()
 {
 ti_copy( o1, i1);
 // copies name, timestamp, samplerate, rangemin,
 // rangemax, unit, scale,
 // but no status
 status = ti_getstatus( i1);
 ti_setstatus( status, o1);
 }

typeinfo_lesen()
 {
 ti_getname( name, i1);
 timestamp = ti_gettimestamp( i1);
 samplerate =  ti_getsamplerate( i1);
 rangemin =  ti_getrangemin( i1);
 rangemax = ti_getrangemax( i1);
 erg = ti_getunit( unit, i1);
 scale =  ti_getscale( i1);
 status = ti_getstatus( i1);
 }
```

```
typeinfo_setzen()
 {
 ti_setname( name, o1);
 ti_settimestamp( timestamp, o1);
 ti_setsamplerate( samplerate, o1);
 ti_setrangemin( rangemin, o1);
 ti_setrangemax( rangemax, o1);
 erg = ti_setunit( unit, o1);
 ti_setscale( scale, o1);
 ti_setstatus( status, o1);
 }

execute
 {
 typeinfo_kopieren();
 typeinfo_lesen();
 typeinfo_setzen();
 // other actions
 }
```

6.2.7 Interpret as a Data Source

If no input is declared with the **trigger** attribute, the Interpret module is called up with every scheduler cycle. This can be used for modelling data sources.

Example 4 (Interpret as a source):

```
output o1("TYPEINFO","TypeInfo","SWORD[]","BIN");
int i;
init
 {
 i=0;
 }
execute
 {
 typeinfo_setzen();
 o1 << i;
 i  = ~i & 1;
 }
```

The example generates a control signal (0 1 0 1 0 ... sequence).

6.2.8 Trigger

Trigger inputs, i.e. inputs for which the **trigger** attribute is specified, are OR-operated as a default. The execute section will be called up if a signal is present at one of the inputs. The TRIGGER = T_AND variable can be used for setting an AND operation as trigger condition. With this variable the module is ready if there are signals at all the trigger inputs.

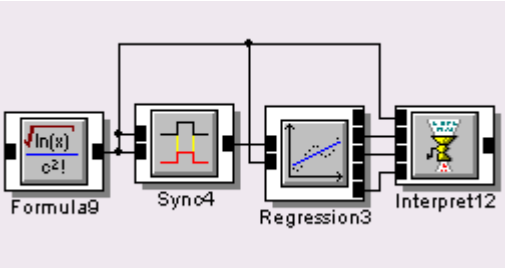
Example 5 (Trigger):

```
input  trigger
i1("TYPEINFO","TypeInfo","DOUBLE[]","TIME_DOMAIN");
input  trigger
i2("TYPEINFO","TypeInfo","DOUBLE[]","TIME_DOMAIN");
output
o1("TYPEINFO","TypeInfo","DOUBLE[]","TIME_DOMAIN");
TRIGGER = T_AND;
int s1; int s2; int i;
execute
 {
 s1 = size( i1);
 s2 = size( i2);
// Intercept runtime error in case of different block
// lengths
 if ( s1 != s2)
  print("Blocklängenfehler");
 else
  {
  typeinfo_setzen();
  for ( i=0; i<s1; i=i+1)
   {
//  In case of incorrect index -> runtime error
   o1 << i1[i] * i2[i];
   }
  }
 }
```

6.2.9 Examples

Linear regression:

The polynominal coefficients of a linear regression y = A0 + A1*x describe a regression straight line through a signal (packet). The y signal is the output of the **Formula** module. **Sync** generates a corresponding time signal x. The coefficients A0, A1, and dx are applied to the inputs of Interpret in addition to the y signal. The output represents the deviation from the straight line.



Pic. 6.2: Regression signal graph

```
input trigger
y(„TYPEINFO","TypeInfo","DOUBLE[]","TIME_DOMAIN");
input trigger
A0(„TYPEINFO","TypeInfo","DOUBLE[1]","TIME_DOMAIN");
input trigger
A1(„TYPEINFO","TypeInfo","DOUBLE[1]","TIME_DOMAIN");
input trigger
dx(„TYPEINFO","TypeInfo","DOUBLE[1]","TIME_DOMAIN");
output
out(„TYPEINFO","TypeInfo","DOUBLE[]","TIME_DOMAIN");
int s;
int i;
execute
 {
 s = size(y);
 ti_copy(out, y);
 for ( i = 0 ;  i < s; ++i )
  {
  out << y[i]- A0[0] / dx[0] + A1[0];
  }
 }
```

Automation of an EMC Measurement:

A measuring set-up should be automated for examining RF interference in the power supply cable of the equipment under test. For this purpose a list of frequency and power values is read in from a file and is output through IEE488-BUS to an oscillator (Hameg HM 8133-2) and an RF power amplifier (Frankonia FLL 75) for control. The amplifier supplies its RF output to a current probe, which is used for inducing currents in the cable that have an interfering effect on the equipment under test. The deviation from the set value of the equipment under test at the respective interference is measured with a multimeter (Keithley 2000). The interference as a function of the frequency is written in a file and displayed graphically.

```
input I_Frq („TYPEINFO", „TypeInfo", „DOUBLE[]",
„TIME_DOMAIN");
input I_Dbm („TYPEINFO", „TypeInfo", „DOUBLE[]",
„TIME_DOMAIN");
input I_Schalter („TYPEINFO", „TypeInfo", „SWORD",
„BIN");
input I_Start („TYPEINFO", „TypeInfo", „SWORD", „BIN");
output O_Frq („TYPEINFO", „TypeInfo", „DOUBLE[]",
„TIME_DOMAIN");
output O_U („TYPEINFO", „TypeInfo", „DOUBLE[]",
„TIME_DOMAIN");

int status;
int s; int l; int t;
double f; double p; double v;
char beffrq[60];
char wertfrq[40];
char befdbm[60];
char wertdbm[40];
char volt_werte[80];
```

```
typeinfo_setzen_f()
 {
 ti_setname(„Frequenz“,O_Frq);
 ti_settimestamp(time(),O_Frq);
 ti_setsamplerate(1.5,O_Frq);
 ti_setrangemin(0.0,O_Frq);
 ti_setrangemax(100000000.0,O_Frq);
 ti_setscale(1.0,O_Frq);
 ti_setstatus(3,O_Frq);
 }

typeinfo_setzen_v()
 {
 ti_setname(„Volt“,O_U);
 ti_settimestamp(time(),O_U);
 ti_setsamplerate(1.5,O_U);
 ti_setrangemin(0.0,O_U);
 ti_setrangemax(1000.0,O_U);
 ti_setscale(1.0,O_U);
 ti_setstatus(3,O_U);
 }
```

```
init
 {
// nitialising the IEEE488 interface (Keithley KPC-488.2)
 initialize (21, 0);
 t=1000000;
 // DMM function V, Range Auto
 send(16,":volt:dc:rang:auto on",status);
// Hameg frequency, attenuation
 send(7,"frq:150000,dbm:-50,am2:80",status);
 }

execute
 {
 if ( I_Start == 1) t = 0;
 // Number of measuring frequencies
 s = size(I_Frq);
 if ( I_Schalter == 1 && t<s)
  {
  // Creates the command for setting the frequency at
  //Hameg HM 8133-2
  beffrq = „frq:";
  f = I_Frq[t];
  ftoa( wertfrq, f, 20);
  strcat( beffrq, wertfrq);

  // Creates the command for setting the attenuation at
  // Hameg HM 8133-2
  befdbm = „dbm:";
  p = I_Dbm[t];
  ftoa( wertdbm, p, 20);
  strcat( befdbm, wertdbm);

  // Sends settings to Hameg HM 8133-2 (IEEE488 equipment
  // adress 7)
  send ( 7, „op1:", status);
  send ( 7, beffrq, status);
  send ( 7, befdbm, status);

  // 1.5 seconds hold on time
  Sleep(1500);
```

```
// Sends settings to Keithley 2000 (IEEE488 equipment /
// address 16)
// Switch off continuous measurement
  send ( 16, „init:cont 0“, status);
  send ( 16, „meas:volt:dc?“, status);
// Read a voltage value (of max. 40 characters) from
// Keithley 2000
// 1 = actual number of characters in string
  enter ( volt_werte, 40, 1, 16, status);
  send ( 16, „init:cont 1“, status); // continuous
// measurement
  v = atof( volt_werte );
  O_Frq<<f;
  O_U<<v;
  typeinfo_setzen_f();
  typeinfo_setzen_v();
  t++;
  }
 }
```

7. Debugger

The graphic debugger in ICONNECT serves for locating faults in the signal graph that occur during the runtime of the program. It can be used for controlling (break, single step) and monitoring (watch window) the data flow. The debugger pauses program execution at fixed positions (breakpoints) and

- allows you to locate faults in the signal graph,
- provides information about the fault position, and
- ensures quick program testing.

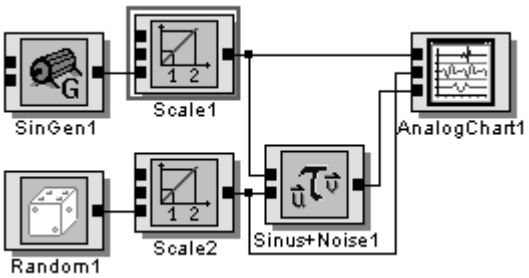
7.1 Breakpoint

The program is stopped at a breakpoint.

Setting a breakpoint:
- Place the mouse pointer on the module in the signal graph
- Press the right mouse button
- Select **Set**/**delete breakpoint** from the pull-down menu

Please note:
A breakpoint can also be set in the running signal graph. It is possible to set an unlimited number of breakpoints. The breakpoints are stored together with the signal graph.

In the signal graph a module with a set breakpoint is marked with a frame (see pic. 7.1). ICONNECT stops before breakpoint execution.



*Pic. 7.1: Signal graph with a breakpoint for the **Scale1** module*

In debugger mode the commands

- **Single step** (Strg+E) Signal graph execution in single steps
- **Continue processing** (Strg+W) Execute signal graph up to the next breakpoint

are available in the measurement menu.

Delete breakpoints | Delete watch widow | Break | Single step | Continue

Pic. 7.2: Section of the toolbar with debug commands

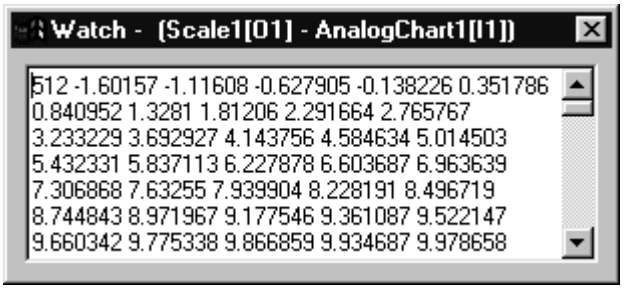
Deleting a breakpoint:

- Place the mouse pointer on the module in the signal graph
- Press the right mouse button
- Select **Set/delete breakpoint** from the pull-down menu, or select **Delete breakpoints** from the **Measurement** menu to delete all the breakpoints.

When you pause the signal graph (menu **Measurement > Break**), you also can execute the signal graph in single steps.

7.2 Watch window

ICONNECT displays the contents of a communication channel in watch windows (see pic. 7.3). By using two watch windows, before and after a breakpoint, you can check the input and output data of a module.



Pic. 7.3: Watch window

Opening a watch window:
- Pause the signal graph
- Double-click on the line to open the watch window

When the signal graph is paused, there is another possibility for displaying the contents of a communication channel. For this purpose position the mouse pointer on the desired line in the signal graph.

The title in the watch window lists the two modules that are connected to each other by means of the watched channel. It is possible to use an unlimited number of watch windows.

Please note:
Watch windows only display data in a paused signal graph.

Deleting watch windows:
Select **Delete watch windows** from the **Measurement** menu to remove all the open watch windows.

9. Appendix

9.1 Data Types in ICONNECT

ICONNECT-data type	C-Data type	Bytes	Range	Resolution
DOUBLE DOUBLE[1] DOUBLE[]	double	8	± 1,7E308	15 digits
FLOAT FLOAT[1] FLOAT[]	float	4	± 3,4E38	7 digits
SDWORD SDWORD[1] SDWORD[]	long	4	-2 147 483 648 ... 2 147 483 647	
UDWORD UDWORD[1] UDWORD[]	unsigned long	4	0 ... 4 294 967 295	
SWORD SWORD[1] SWORD[]	int (32 Bit)	4	-2 147 483 648 ... 2 147 483 647	
UWORD UWORD[1] UWORD[]	unsigned int	4	0 ... 4 294 967 295	
UBYTE UBYTE[1]	unsigned char	1	0 ... 255	
UBYTE[]	CString			

9.2 Modules in Connection with External Data Sources

Module name	Properties
LoadTable / **SaveTable**	max. 16 columns (=16 inputs); no time behaviour, but data transfer as fast as possible (for Ascii, Excel, etc.);
LoadAscii / **SaveAscii**	ICONNECT data format with time and unit Two channels (x,y or x,t); optionally incl. time behaviour, i.e. during loading data are transferred with the same speed they were originally saved with.
DBLoad / **DBSave**	Connection to database (Access)
FileDlg	Allows interactive input of file name and path (for LoadTable, LoadAscii, and DBLoad).

9.3 Error Messages

• Are the following type declarators compatible?
This is not an error, but only a notice: The data types are compatible, but the type declarators are different. Since type declarators may also be assigned manually, the user (signal graph developer) can decide whether a combination of these inputs/outputs makes sense.

• Arithmetic error
This error occurs in case of a division by 0, or if the result of a calculation exceeds the value range (e.g. if an attempt is made to encode 4096 with 12 bits).

• Block lengths at the inputs do not correspond
The upstream modules of a module with several inputs supply data blocks with differing numbers of data. For correct processing these must be identical.
Remedy: Shortening of blocks can be achieved, for example, with the Resampling module (data reduction).
Synchronisation of data is possible with the **Sync** module.

• Buffer overflow
The data buffer used by this module was not able to store all data. This occurs if the data buffer is too small, or if data processing is too complex.

• Different base types
(Data types not compatible)
These modules cannot be connected with each other.
Possible solution: Type conversion with the TypeCast module often allows connection despite different data types. This cannot be recommended, because depending on the type of conversion there will be rounding errors, complete data sectors will be cut, or a conversion of this kind simply makes no sense.

• Different number of validation parameters! 2 vs. 1.
A port that requires several parameters cannot be connected with a port that only supplies one parameter.

• Different sampling rates at the inputs
The module requires the same Sample Rate at several inputs (consult the respective online help on the module).
The sampling rates can be visualized and compared with the TIDisp module.

• Driver error:
A virtual device driver (VxD) cannot operate correctly because, for example there is no supported hardware (A/D converter card, etc.), or because the set I/O addresses, DMAs, IRQs, etc. are invalid or already used.
Remedy: Check the hardware (damaged, incorrectly installed), adapt the driver settings to the system configuration of the computer (Device manager > Settings in the system control panel).

• Error during parameter transfer via the EXT input.
There are invalid parameters at the EXT input.
Remedy: Reconfigure the module that supplies the parameters.

• Error during parameter transfer via the EXT input:
The parameters that are supplied to the module through the EXT input do not correspond with the expected format. If the ParamConv module is the parameter source, this must be reconfigured (see ParamConv.hlp).
This error message may occur if the parameters exceed certain limits. (Example: The lower limit for the representation of a sine signal is 2, or the upper limit lies below the lower limit)

• File error:
A file could not be opened or saved. Possible causes: The file does not exist, is not at the specified location (path), is damaged, has the incorrect format (old version), is write-protected, or the storage medium (hard disk) does not offer enough free space.

• Incorrect ICONNECT version
The signal graph that should have been loaded has another version number that is not compatible with this ICONNECT version.

• Module xxx signals an error during initialisation
An attempt was made to use modules belonging to the ICONNECT runtime version with an ICONNECT developer version, or vice versa.
Remedy: Create the module path to the correct modules. Otherwise reinstall ICONNECT.

• No ICONNECT signal graph
The selected document does not contain a valid signal graph, or it is damaged.

• Number of blocks does not correspond
The upstream modules of a module with several inputs supply different numbers of blocks. The module operates with data blocks, therefore the same number of blocks must be supplied.

• Obligate input not connected
An input the module needs for processing is not connected.
Remedy: Check the connection, consult the corresponding help file on the module.

• Packet jam
This occurs if the upstream module supplies data packets faster than they can or may be processed by this module.
It also occurs if a module has several upstream modules that supply parallel data packets in different time intervals.
Remedy: Excess packets can be skipped with the Sync module (loss of data), and a packet jam can be avoided.

• Packet status does not correspond
Cause: Data packets arriving asynchronously (Example: One input receives packets with several blocks, while another input of the same module only receives data packets with one block).
Possible remedy: Synchronisation with the Sync module.

• The file is no DisplayManager document
The selected document does not correspond with the required format and was therefore not loaded.
Possible cause: A DisplayManager window was active when you tried to load a signal graph (*.tc2) or another file. An active output window of a DisplayManager only allows loading of its own special file format.
The window of the signal graph must be activated first in order to be able to load a signal graph.

• The resource ID of a module is used twice
A module exists several times under different names in the module directory.
Remedy: Delete the modules and reinstall the program.

• The signal graph contains a recursive nesting of macros
Recursive nesting of macros makes no sense. A corresponding signal graph cannot be started.

• The signal graph could not be opened
Check whether all the required modules are present.
Possible cause: Module path incorrectly set, modules missing, or not licensed for this dongle.

• The status of two inputs does not correspond:
This occurs if upstream modules supply data packets asynchronously, but the module processes the data in packets. Remedy is possible with the Sync module (synchronisation of data).

• The time stamp on channel I1 is incorrect! Please check your signal graph.
A module that processes data with the help of the time signal (=timestamp) receives an invalid timestamp.
Troubleshooting: The time signal can be checked with the TIDisp module. If the module supplies no time signal or an invalid time signal, TIDisp indicates "-" as the timestamp.
Remedy: Reconfigure the module that generates the incorrect time signal, or replace it with another module.

• The units do not match:
It makes no sense to add or subtract data with different units.

• There are no source modules in the signal graph
A signal graph without a data source makes no sense and can therefore not be started.

• This module can not be used in the demo-version
Certain modules can only be used in the licensed version. Please contact your distributor for licensing.

• This input of the module is already in use
Inputs can only receive data from one data source.

• This channel is already used by another transmitter
Two Communicate modules cannot write into the same channel.
Please select another channel.

• Unable to allocate enough memory
This may be caused by too many open applications,
too little memory (RAM), or by a hard disk with insufficient capacity. It is also possible that modules allocate a lot of memory, if the buffer sizes are exaggerated.
Remedy: Fist close all the applications that are not required (Word, etc.). The virtual memory should be located on a hard disk that offers enough free space (64 MB minimum). If this is not enough, it is recommendable to install a larger hard disk and maybe more memory.
Please contact your system / signal graph developer, if this problem occurs repeatedly.

• Unable to generate module
Causes: The module is not licensed, is damaged, or has the incorrect version.
Remedy: If you use a demo-version, reinstall it. Otherwise license the module (please contact your distributor).

• Validation failure. Different types
These modules can never be connected with each other. A connection of these modules never makes sense.

• Warning. The signal graph contains a cycle! (It can only be aborted using the `Kill`button).
This is no error, but if the cycle does not contain an abort condition, the signal graph can only by stopped with an emergency stop (button/menu entry).